STORAGE TECHNIQUES IN FLASH MEMORIES AND PHASE-CHANGE MEMORIES

A Dissertation

by

HAO LI

Submitted to the Office of Graduate Studies of
Texas A&M University
in partial fulfillment of the requirements for the degree of

DOCTOR OF PHILOSOPHY

August 2010

Major Subject: Computer Engineering

STORAGE TECHNIQUES IN FLASH MEMORIES AND PHASE-CHANGE MEMORIES

A Dissertation

by

HAO LI

Submitted to the Office of Graduate Studies of
Texas A&M University
in partial fulfillment of the requirements for the degree of

DOCTOR OF PHILOSOPHY

Approved by:

Chair of Committee,	Anxiao Jiang
Committee Members,	Jianer Chen
	Jennifer Welch
	Alexander Sprintson
Head of Department,	Valerie E. Taylor

August 2010

Major Subject: Computer Engineering

ABSTRACT

Storage Techniques in Flash Memories
and Phase-change Memories. (August 2010)
Hao Li, B.S., Tsinghua University;
M.S., Chinese Academy of Sciences
Chair of Advisory Committee: Dr. Anxiao Jiang

Non-volatile memories are an emerging storage technology with wide applications in many important areas. This study focuses on new storage techniques for flash memories and phase-change memories. Flash memories are currently the most widely used type of non-volatile memory, and phase-change memories (PCMs) are the most promising candidate for the next-generation non-volatile memories. Like magnetic recording and optical recording, flash memories and PCMs have their own distinct properties, which introduce very interesting data storage problems. They include error correction, cell programming and other coding problems that affect the reliability and efficiency of data storage. Solutions to these problems can significantly improve the longevity and performance of the storage systems based on flash memories and PCMs.

In this work, we study several new techniques for data storage in flash memories and PCMs. First, we study new types of error-correcting codes for flash memories – called error scrubbing codes –that correct errors by only increasing cell levels. Error scrubbing codes can correct errors without the costly block erasure operations, and we show how they can outperform conventional error-correcting codes. Next, we study the programming strategies for flash memory cells, and present an adaptive algorithm that optimizes the expected precision of cell programming. We then study data

storage in PCMs, where thermal interference is a major challenge for data reliability. We present two new coding techniques that reduce thermal interference, and study their storage capacities and code constructions.

ACKNOWLEDGMENTS

Thanks to Prof. Anxiao (Andrew) Jiang, my research adviser, who offered me the opportunity to work with a group of talented and energetic people. Andrew showed me the art and science in research in computer systems. I am really impressed by Andrew's intuition, thoughtfulness, and quick comprehension. His valuable suggestions and the discussions on research ideas have led me toward better and clearer understanding of my research subjects. I greatly cherished his advice on research, writing, and presentation. Without Andrew's encouragement and help during the difficult times in my PhD study, this dissertation would not have been possible.

I would like to thank Dr. Jennifer Welch, Dr. Jianer Chen, and Dr. Alex Sprintson for serving on my PhD committee, and Dr. Yoonsuck Choe for attending my defense. I have had a deeper understanding of the subject thanks to their suggestions.

I would also like to thank my colleagues Fenghui Zhang, Yue Wang, Vishal Kapoor, and Shoeb Ahmed Mohammed, not only for their cooperative works and helping hands during these school years, but also for the friendly and relaxing working environment created by them.

Last, but not the least, I would like to extend my appreciation to my family and all my friends for their long-term caring and support.

TABLE OF CONTENTS

CHAPTER Page

LIST OF TABLES

LIST OF FIGURES

CHAPTER I

INTRODUCTION

The work in this thesis focuses on new storage techniques for flash memories and phase-change memories (PCMs). They are two key members in the family of non-volatile memories (NVMs). In this chapter, we first introduce flash memories and phase-change memories, and point out their challenges. We then introduce the contribution of our work. We also present an overview of the related areas.

A. Two Key Non-volatile Memory Technologies

Non-volatile memories (NVMs) are computer memories that can retain the stored information even when they are not powered. Examples of non-volatile memories include hard disks, floppy disks, magnetic tapes, CMOS chips, CDs, DVDs, read-only memories, flash memories and phase-change memories. Conventionally, non-volatile memories are typically used for long-term persistent storage. For example, in computers, non-volatile memories are used to store essential system information and data (BIOS), and as secondary storage devices (hard disks, CDs, etc.). In addition, non-volatile memories have also been widely used in mobile devices, such as cell phones, digit cameras, palms, CD/DVD players, etc.

Non-volatile memories can be categorized by electrically addressed systems (e.g., read-only memories, flash memories and phase-chase memories) and mechanically addressed systems (e.g., hard disks, optical disc, and magnetic tapes). Traditionally, electrically addressed systems are expensive but fast, whereas mechanically addressed systems have lower prices per bit and larger storage capacities but are slow. How-

---

The journal model is *IEEE Transactions on Automatic Control.*

ever, the recent technology development has made some new electrically addressed memories – such as flash memories and phase-change memories – to have both low prices and high capacities, while still maintaining their fast reading/writing speed.

Flash memories were first invented by Dr. Fujio Masuoka at Toshiba in 1980. In recent years, flash memories have become the most widely used type of non-volatile memories. Flash memories are close to an "ideal" memory in the sense that they can be electrically erased and programmed in-system, offer high storage density and low cost-per-bit, and have the random access capability, bit alterability, short read time and excellent reliability. Because of these advantages, flash memories have been widely used in numerous applications, including PDAs (personal digital assistants), laptop computers, digital audio players, digital cameras and mobile phones. In 2008, flash memories accounted for over $23 billion in sales, which was about 8 percent of the total $277 billion sales of the semiconductor industry. And they are expected to increase at 21 percent annually, higher than average growth rate of 18 percent of the semiconductor industry [6].

A flash memory cell has $q \geq 2$ levels – level $0, 1, \cdots, q-1$ – which can be increased or decreased by charge injection or removal based on the Fowler-Nordheim tunnelling mechanism or the hot-electron injection mechanism [4]. To increase data density, multi-level cells (MLCs) with greater values of q are actively developed. Flash cells are organized as blocks, each containing about 10^5 cells. Although it is comparatively simple to increase a cell level, it is very difficult to decrease one. To decrease any cell level, the whole block of cells must be erased and then reprogrammed. Block erasures significantly reduce the longevity, reliability and speed of flash memories [4]. To minimize the number of block erasure operations, many techniques in conventional storage systems need to adapt when we apply them to flash memories. A basic rule for adaptation is to realize the change of data by only increasing the cell levels, thus

avoiding block erasures. In this thesis, we study how to use this rule while addressing various problems in flash memories.

Phase-change memories (PCMs) are one of the most promising candidates for the next-generation NVMs. Compared to the widely used flash memories, PCMs can potentially scale to much smaller cell sizes and achieve higher storage capacities. They can also have substantially better endurance, data retention and read/write speed [3]. However, scaling down cell sizes can also bring significant challenges, and solving them will be key to the PCM development [3].

The basic storage unit of a phase-change memory – a PCM cell – has at least two states: the *amorphous state* and the *crystalline state*. To achieve higher storage capacity, multi-level cells (MLCs) are being developed, where additional *partially crystalline states* are used [3]. We model the $q \geq 2$ states of a PCM cell by q levels – levels $0, 1, \ldots, q-1$ – where level 0 is the amorphous state, level $1, \ldots, q-2$ are the partially crystalline states, and level $q-1$ is the crystalline state. As a cell becomes more crystallized, its level increases.

The level of a PCM cell is switched using high temperatures. A cell can be heated by a high cell-melting temperature (about $600^oC \sim 700^oC$) to change to level 0 (amorphous state), or be heated by a more moderate temperature to increase its level (i.e., to a more crystallized state). To model the direct switching of states, we use the diagram in Fig. 1 (for $q = 4$ as an example) [16]. We see that the cell can be changed from any level $i \in \{1, 2, \ldots, q-1\}$ directly to level 0 (called a *RESET* operation), and from any level i directly to level j for $0 \leq i < j \leq q-1$ (called a *SET* operation). However, for $0 < j < i \leq q-1$, to change it from level i to level j, both the RESET and SET operations are needed.

A major and widely acknowledged challenge for PCM is the thermal issue, because the amorphous and partially-crystalline states are only semi-stable states, and

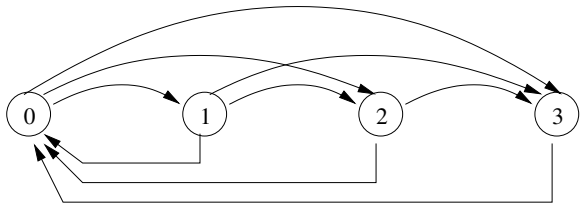

Fig. 1. The transitions among q cell levels of a phase-change memory (PCM) cell. (Here $q = 4$.) The forward and backward edges represent the SET and RESET operations, respectively.

high environmental temperatures can further crystalize the cell, i.e., unintentionally increase the cell level [3, 19]. In this thesis, we study two types of codes for addressing this problem.

The rest of the chapter is organized as follows. In sections B and C, we introduce the main challenges facing flash memories and phase-change memories. In section D, we introduce the topics of our work and its contributions. In section E, we survey some related areas.

B. Challenges for Flash Memories

In this section, we first introduce the principles and operations in flash memories, and then discuss the resulting challenges. To meet these challenges, we will study two important storage technologies for flash memories, namely *error correction* and *cell programming*.

1. Principles and Operations in Flash Memories

Flash memories are a type of EEPROM (Electrically Erasable Programmable Read-Only Memory). A flash memory cell resembles a standard MOSFET (metal-oxide-semiconductor field-effect transistor), except that the transistor has two gates instead of one [4]. On the top is the control gate (CG) (as in other MOS transistors), but

below it there is a floating gate (FG) insulated all around by an oxide layer. The electrons that the floating gate picks up will be trapped there for many years under normal conditions. The trapped electrons can significantly affect the threshold voltage needed to turn on the transistor (i.e., for the conducting state). With no electron in the floating gate, the threshold voltage will be low, and the transistor will be turned on easily, whereas with electrons injected into the floating gate, the threshold voltage will become higher.

Suppose that the transistor's being on corresponds to the state "0", and the transistor's being off corresponds to the state "1". By checking if the transistor is on or off, the amount of charge trapped in the floating gate, called the *cell level*, can represent one bit of information (per cell). When reading a multi-level cell, the amount of current flowing through the transistor is sensed (rather than simply checking its presence or absence), which can be used to determine more precisely the amount of charge in the floating gate. The level of a multiple-level cell can represent more than one bit.

a. Read Operation

In order to determine a cell's level, a certain reading voltage is applied to the transistor, and the amount of current flowing through the transistor is sensed. In NOR flash memories, a cell's level can be read individually, whereas in NAND flash memories, the cells in the same page must be read together. The reading procedure is relatively simple and fast in both types of flash memories.

b. Programming Operation

A flash cell can be programmed (i.e., setting to a higher level) by applying a high voltage on the control gate to inject electrons into the floating gate. The cells of both

NOR and NAND flash memories can be programmed relatively fast.

c. Erase Operation

To erase a flash cell (i.e., resetting it to the "0" state), a high voltage of the opposite polarity is applied to the control gate, pulling the electrons off the floating gate through *quantum tunneling* [4]. The erase operation can only be performed on a block-wise basis, i.e., all the cells in a block must be erased together. A cell block usually consists of 1,000 to 1,000,000 cells.

Note that the block erasure is a fairly strenuous process, which leads to the flash memory's most debilitating limitation: the limited number of erasure operations the cells can endure. Every time the system erases a cell, it slightly damages the insulating barrier. Eventually, the cell becomes useless. The typical lifetime of a flash memory block is about $10^4 \sim 10^5$ program-erasure cycles.

d. Decreasing Cell Levels

Decreasing a cell level is implemented by the combination of erasing and programming. As a toy example, suppose we want to decrease the first cell's level from "2" to "1" in a block of 4 multi-level cells, whose current cell levels are "2101". We first need to erase the block to "0000", and then program each cell until their levels reach "1101". A typical block consists of many cells. So the speed of decreasing a cell level can be much slower than increasing cell levels. Also, decreasing cell levels leads to block erasures, which shorten the longevity of flash memories. So it will be beneficial to avoid it if possible. In our work we will realize the change of data by only increasing cell levels, not decreasing them, and thus achieving better performance.

2. Data Reliability in Flash Memories

After cells are programmed, the stored data can be affected by various noise mechanisms, including charge leakage, read disturbance, write disturbance, etc. [4]. So it can be important to store data with a strong error-correcting code. In addition, a common technique in store systems called *memory scrubbing* [23] actively and periodically removes errors: When the errors occur in codewords, the memory-scrubbing procedure decodes the codewords and writes their correct values back into the memory. However, for flash memories, since the cell levels often have to be decreased when writing back the error-free codewords, the conventional memory scrubbing procedure can be very costly. The challenge will be addressed by the *error-scrubbing code* technique in our research.

3. Cell Programming in Flash Memories

When programming a cell, the charge (e.g., electrons) is injected into the cell, and the injected charge becomes trapped. The amount of charge in a cell determines its level. Overshooting is very costly for programming because once the injected charge overshoots the target level, the block needs to be erased and then reprogrammed. In the industry, when a cell is being programmed, the cell level is only allowed to increase [4]. The charge-injection process is noisy, so usually multiple rounds of charge injection are used to shift the cell level monotonically and cautiously toward the target level [1].

It is interesting to study how to program cells accurately, since the precision of cell programming determines the storage capacity of flash memories [11]. We will study cell programming strategies for two data representation schemes in flash memories, namely, the multi-level cell technology and the rank modulation technology.

C. Challenges for Phase-change Memories

The PCM is a non-volatile memory that can switch between at least two phases, the *amorphous* phase and *crystalline* phase. A cell in the amorphous phase has high electrical resistivity, whereas a cell in the crystalline phase exhibits a lower resistivity, generally with a ratio of 1000 times. Because of the significant contrast between the amorphous and crystalline phases, a PCM cell can easily store one bit of information. Recently, cells with two *partially crystalline phases* have been invented for storing two bits per cell. To SET the cell into the crystalline phase, an electrical pulse is applied to heat the cell above the crystallization temperature. In the RESET operation, a larger and shorter electrical current is applied to melt the cell. Then molten cell will quench into the amorphous phase. Note that the melt temperature in the RESET operation is much higher than the crystalline temperature in the SET operation.

We consider two potential thermal-based challenges when the PCM's cell density scales toward its limit. The first one is the thermal crosstalk problem, namely, when a cell is RESET (to level 0) by the high melting temperature, the heat affects its adjacent cell and makes it further crystalized [3, 19]. Note that this may happen both when the adjacent cell is not being programmed and when it is being SET, unless it is already in the fully crystalized state (level $q-1$). It is because in both cases, the semi-stable cell state is sensitive to high temperatures.

The second problem is the local thermal accumulation problem. When cells are repeatedly programmed, the heat can accumulate in the area [19]. This residual heat can be a major factor that limits the writing bandwidth of PCMs, because the writing accuracy is sensitive to temperature [3, 19]. When the cell density scales toward its limit, relative to the high I/O speed, it can take nontrivial time for the locally generated heat to spread out uniformly in the memory chip. So if an application

repeatedly writes a cluster of adjacent cells at very high speed, the accumulated heat may appear localized [19]. In that case, it is worth considering whether there exist schemes that can make the thermal accumulation more balanced.

The above problems have been considered in [3, 19] from the device and system perspectives. In this thesis, we consider coding techniques for these potential challenges. For the thermal crosstalk problem, we use a scheme that removes the crosstalk interference, and then study coding techniques that reduce the programming cost (measured by the number of RESET operations). For the local thermal accumulation problem, we study coding techniques that impose time and space constraints on writing, to help the heat generated by programming be more balanced spatially.

D. Contributions of This Work

In this work, we address the challenges facing flash memories and phase-change memories by three techniques: *error-scrubbing codes* for repeatedly and efficiently removing errors from flash memories, *optimized cell programming* for accurately programming flash memory cells, and *constrained codes* for the reliable programming and storage of PCM cells. In the following, we introduce the three topics.

1. Error Scrubbing Codes for Flash Memories

A prominent property of flash memories is that although it is easy to increase a cell level, to decrease any cell level, a whole block of cells have to be erased and reprogrammed. To minimize the number of expensive block erasure operations and to maintain the data integrity, the data needs to be stored with a strong error-correcting code that can correct enough errors between two erasure operations.

We develop the concept of *error scrubbing codes* to adaptively scrub the memory (i.e., decode codewords and write the correct values back into the memory). With this new type of error-correcting codes, the cell levels are actively increased when errors appear, even if the errors increase cell levels as well. The implementation is simple: the memory constantly reads the cells of a codeword; if a new error is detected, the memory increases the cell levels to a new state. No block erasure is needed unless the cell levels have reached the top. The key idea of error-scrubbing codes is that through the active adjustment of cell levels, which we call *scrubbing*, we can reduce the number of states that a given number of errors can turn the cells into, thus allowing the packing of more codewords for a higher rate. We show that the performance of error-scrubbing codes can exceed that of conventional codes. We present two families of code constructions based on the L_1 metric and a modular technique, respectively, and show their asymptotic optimality.

2. Optimized Cell Programming for Flash Memories

Cell programming is the process of increasing a cell's level to the target value via charge injection, and the storage capacity of flash memories is limited by the precision of cell programming. To optimize the precision of the final cell level, a cell is programmed adaptively with multiple rounds of charge injection. Due to the high cost of block erasure, when cells are programmed, their levels are only allowed to increase. It is interesting to study how well such storage media can be programmed.

We focus on the programming strategy that optimizes the expected precision. The performance criteria considered here include two metrics that are suitable for the multi-level cell technology and the rank modulation technology, respectively. Assuming that the charge-injection noise has a uniform random distribution, we present an effective algorithm for finding the optimal programming strategy. The optimal

strategy can be used to program cells efficiently.

3. Constrained Codes for Phase-change Memories

A PCM's cell states are switched using high temperatures. As the semi-stable states of PCM cells are sensitive to temperatures, scaling down cell sizes can bring significant challenges. We consider two potential thermal-based interference problems, the thermal crosstalk problem and the local thermal accumulation problem, and propose new constrained codes for solving them.

To address the thermal crosstalk problem, we present the *symbol-constrained codes*. In a symbol-constrained code, when a codeword is written, the cells that go through the RESET process have a limited run-length. We show that symbol-constrained codes can reduce the number of resets for rewriting data and extend the longevity of the PCM.

To address the local thermal accumulation problem, we present space-time constrained codes. A space-time constrained code limits the number of times a cell can be programmed consecutively, and limits adjacent cells from being programmed together. The goal is to reduce the heat accumulated in the local areas of the PCM. We study the capacity of the constrained codes, and present novel code constructions.

E. Related Work

The work in this thesis relates to a number of important research areas. They include error-correcting codes, the memory scrubbing technology, constrained codes, and information representation and coding for flash memories. We briefly survey these areas in this section.

1. Error-correcting Codes and Memory Scrubbing

The general definition of error correction is to detect errors and reconstruct the original error-free data. An error-correcting code (ECC) is a system of adding redundant data (parity data) to a message such that it can be recovered by a receiver even when a number of errors (up to the capability of the code) are introduced. ECCs are usually categorized into convolutional codes and block codes. Convolutional codes work on bit or symbol streams of arbitrary length, while block codes are processed on fixed-size blocks. Examples of block codes include repetition codes, Hamming codes, Reed-Solomon codes, etc. An ECC contains two processes, *encoding* and *decoding*. A good ECC should correct as many errors as possible and has as many codewords as possible. However, there is a tradeoff between them. There has been lots of work on error-correcting codes, and the error correction technology has been pivotal to the development of storage systems [17, 20, 21].

Memory scrubbing is the process of detecting and correcting errors in memories by using error-correcting codes [23]. Systems using memory scrubbing check the memories periodically, correct errors and write the correct data back into the memory immediately after errors are detected. If the memory is checked frequently enough, memory scrubbing can effectively prevent the accumulation of errors in the codewords, and thus ensure data reliability.

It is very costly to use conventional error-correcting codes for memory scrubbing in flash memories because of the block erasures they would trigger. This has motivated our study of error-scrubbing codes, which balance the error-correction capability of the codes and the cost of block erasures.

2. Constrained Codes

Constrained coding has been a critical technology for the development of magnetic recording and optimal recording. A well known type of constrained codes is the run-length-limited (RLL) code, where the run-length of consecutive zeros in the codeword is constrained to help synchronization during reading and reduce inter-symbol interference. Generally speaking, a constrained code refers to a constrained encoder and a constrained decoder. The encoder transforms arbitrary input data sequences into sequences (codewords) that satisfy the given constraint. The decoder recovers the input data from codewords. The purpose of adding the constraints is to improve the system's performance by constraining the codewords in such a way as to reduce the likelihood of errors. Different from conventional error-correcting codes, where the distance between codewords is the major criterion for the error-correction performance, in constrained codes the performance is measured by properties of the individual codewords. The constraint can often be described by a labelled graph (called its graph representation), and the labels of a path in the graph form a valid codeword. There has been lots of study on the capacity of constrained codes and code constructions [18]. The new constrained codes we study for PCMs have different forms compared to all the existing constrained codes.

3. Information Representation for Flash Memories

Based on the unique properties of flash memories, new data representation schemes have been developed in recent years. Notably, they include codes for rewriting data and the rank modulation scheme [14].

Rewriting codes are schemes that allow data in a flash memory to be rewritten many times without block erasures. The flash-memory model commonly used for

rewriting codes is the *Write Asymmetric Memory* (WAM) model [9],in which each cell's level can only increase, not decrease. WAM is a special case of the *generalized write-once memory* (WOM) model, which allows the state transitions of cells to be any acyclic directed graph [7, 8, 22]. Examples of rewriting codes include *floating codes* [9, 10, 12], *trajectory codes* [12], *buffer codes* [2], etc. A floating code jointly stores multiple variables in flash memory cells, and generalizes the definition of WOM codes [22]. A number of floating code constructions have been presented, some of which are proved to be optimal or asymptotically optimal [9]. Furthermore, the model in floating codes has been generalized by using directed graphs to represent how rewrites may change the stored data [12]. For this generalized rewriting model, the trajectory codes have been presented, and proved to be asymptotically optimal [12]. Different from floating codes and trajectory codes, buffer codes record the most recent values of a variable, and some code constructions have been presented [2].

Rank modulation is a scheme that uses the relative order of the cell levels, instead their absolute values, to represent data [13, 15]. The n cells in a group can introduce $n!$ possible rank permutations, and store up to $\log_2 n!$ bits of information. To write a permutation, we program the cells from the lowest level to the highest level. This writing procedure removes the risk of charge overshooting and makes cell programming reliable. Rewriting codes and error-correcting codes for rank modulation have been presented [13, 15]. In our research on cell programming, we study the programming strategy for rank modulation, and present an optimal algorithm.

The rest of the thesis is organized as follows. In Chapter II, we present two families of error-scrubbing codes for flash memories. In Chapter III, we study an optimized cell programming scheme for flash memories. In Chapter IV, we present two types of constrained codes for phase-change memories. In Chapter V, we summarize the results, and discuss some open problems.

CHAPTER II

ERROR SCRUBBING CODES

The motivation for error scrubbing codes arises from the fact that the data stored in flash memories is not always reliable. The stored data can be lost due to charge leakage, a long-term factor that causes the data retention problem. The data can also be affected by other mechanisms, including read disturbance, write disturbance [4], etc. All these mechanisms change cell levels and can cause errors in flash memories.

To maintain the data integrity, the data needs to be stored with a strong error-correcting code that can correct enough errors between two erasure operations. While errors may accumulate in the codewords, with the next block erasure, the codewords can be decoded and the original error-free codewords can be written back into the block. This is called *memory scrubbing*, a commonly used operation in storage systems [23]. However, although memory scrubbing works very successfully for previous storage systems, it faces a significant challenge for flash memories: the block erasures triggered by memory scrubbing can substantially reduce the longevity, speed and efficiency of the flash memories. As flash memories scale toward higher data densities, the cost of block erasures will become even higher. Therefore, a new coding scheme is need for flash memories that can balance well the error correction capability of the codes and the block-erasure cost caused by memory scrubbing.

For this purpose, we introduce the concept of *error scrubbing codes*. With this new type of error-correcting codes, the cell levels are actively increased when errors appear, even if the errors increase cell levels as well. The implementation is simple: the memory constantly reads the cells of a codeword; if a new error is detected, the memory increases the cell levels to a new state. No block erasure is needed unless the cell levels have reached the top. The key idea of error-scrubbing codes is that

through the active adjustment of cell levels, which we call *scrubbing*, we can reduce the number of states that a given number of errors can turn the cells into, thus allowing the packing of more codewords for a higher rate. In this report, we show that the performance of error-scrubbing codes can exceed that of conventional codes by presenting two families of code constructions based on the L_1 metric and a modular technique, respectively, and show their asymptotic optimality.

A. Notations

Let $c_1, c_2, \cdots, c_n$ denote the levels of n cells of q levels. Here $c_i \in \{0, 1, \cdots, q-1\}$ denotes the i-th cell's level, for $i = 1, 2, \cdots, n$. The vector $\vec{c} = (c_1, c_2, \cdots, c_n)$ is called the *cell state*. Let $\mathcal{S}_{n,q}$ denote the set of all q^n cell states. Given two cell states $\vec{c}_A = (c_1, c_2, \cdots, c_n)$ and $\vec{c}_B = (c'_1, c'_2, \cdots, c'_n)$, if $\forall\ i$ we have $c_i \geq c'_i$, we denote it by $\vec{c}_A \geq \vec{c}_B$ and say that $\vec{c}_A$ is above $\vec{c}_B$. If $\vec{c}_A \geq \vec{c}_B$ and there exists some i such that $c_i > c'_i$, we denote it by $\vec{c}_A > \vec{c}_B$ and say that $\vec{c}_A$ is strictly above $\vec{c}_B$. Here we consider codes that correct errors between two block erasures, so the memory controller can only increase cell levels, not decrease them [9]. However, the errors (i.e., noise) may both increase and decrease cell levels, unless they are asymmetric errors.

An *error set* ε is a set of integral vectors of length n. An *error* is a vector in the set ε. For convenience, we always assume that $(0, 0, \cdots, 0) \in \varepsilon$. Give an error $\vec{e} = (e_1, e_2, \cdots, e_n) \in \varepsilon$, it can change a cell state $\vec{c} = (c_1, c_2, \cdots, c_n)$ to $\vec{c} + \vec{e} = (c_1 + e_1, c_2 + e_2, \cdots, c_n + e_n)$. For example, if ε consists of those errors that can increase some cell level by one, then $\varepsilon = \{(e_1, e_2, \cdots, e_n) \mid \sum_{i=1}^{n} e_i \leq 1,\ e_i = 0 \text{ or } 1 \text{ for all } i\}$.

A *scrubbing function* is a mapping $f : \mathcal{S}_{n,q} \to \mathcal{S}_{n,q}$ such that $\forall\ \vec{c} \in \mathcal{S}_{n,q}$, $f(\vec{c}) \geq \vec{c}$. The idea of the error-scrubbing code is that when the cell state is $\vec{c}$, the memory will update it to $f(\vec{c})$ by charge injection.

Let $t \geq 1$ be an integer. Let $\vec{c} \in S_{n,q}$ be a cell state, and let $\vec{e}_1, \vec{e}_2, \cdots, \vec{e}_t \in \varepsilon$ be t errors. When the first error $\vec{e}_1$ changes the cell state from $\vec{c}$ to $\vec{c}+\vec{e}_1$, the memory will update the cell state to $f(\vec{c}+\vec{e}_1)$. When the second error $\vec{e}_2$ changes the cell state to $f(\vec{c}+\vec{e}_1)+\vec{e}_2$, the memory will update the cell state to $f(f(\vec{c}+\vec{e}_1)+\vec{e}_2)$. And so on. In general, for $i = 1, 2, \cdots, t$, define a function $g(\vec{c}; \vec{e}_1, \cdots, \vec{e}_i)$ as follows: $g(\vec{c}; \vec{e}_1) = \vec{c}+\vec{e}_1$; for $i = 2, \cdots, t$, $g(\vec{c}; \vec{e}_1, \cdots, \vec{e}_i) = f(g(\vec{c}; \vec{e}_1, \cdots, \vec{e}_{i-1})) + \vec{e}_i$. Then, when the initial cell state is $\vec{c}$ and t errors $\vec{e}_1, \vec{e}_2, \cdots, \vec{e}_t$ sequentially appear, the memory will sequentially change the cell state to $f(g(\vec{c}; \vec{e}_1)), f(g(\vec{c}; \vec{e}_1, \vec{e}_2)), \cdots, f(g(\vec{c}; \vec{e}_1, \cdots, \vec{e}_t))$. (By default, we assume that $g(\vec{c}; \vec{e}_1), g(\vec{c}; \vec{e}_1, \vec{e}_2), \cdots, g(\vec{c}; \vec{e}_1, \cdots, \vec{e}_t)$ all belong to $\mathcal{S}_{n,q}$.) The set of cell states $trace(\vec{c}; \vec{e}_1, \cdots, \vec{e}_t) = \{\vec{c}\} \cup \{g(\vec{c}; \vec{e}_1, \cdots, \vec{e}_i) \mid i = 1, 2, \cdots, t\} \cup \{f(g(\vec{c}; \vec{e}_1, \cdots, \vec{e}_i)) \mid i = 1, 2, \cdots, t\}$ are called the *trace* caused by the initial cell state $\vec{c}$ and the t errors $\vec{e}_1, \vec{e}_2, \cdots, \vec{e}_t$.

We are now in a position to define error-scrubbing codes.

Definition 1. Error-Scrubbing Code. *Let $\mathcal{C} \subseteq \mathcal{S}_{n,q}$ be a subset of cell states. Let $t \geq 1$ be an integer. Every vector in $\mathcal{C}$ is called a codeword. For every codeword $\vec{c} \in \mathcal{C}$, the set of cell states $B_{\vec{c}} = \bigcup_{\vec{e}_1, \cdots, \vec{e}_t \in \varepsilon} trace(\vec{c}; \vec{e}_1, \cdots, \vec{e}_t)$ is called the* "decoding sphere" *of $\vec{c}$. Every vector in $B_{\vec{c}}$ is decoded as the data represented by the codeword $\vec{c}$. Then, $\mathcal{C}$ is called a* t-error-scrubbing code *if $\forall$ $\vec{c}_1$ and $\vec{c}_2$ in $\mathcal{C}$, $B_{\vec{c}_1} \cap B_{\vec{c}_2} = \emptyset$.*

It is simple to see that when data are stored as codewords of a t-error-scrubbing code, any sequence of t or fewer errors can be corrected. For a t-error-scrubbing code $\mathcal{C}$, we define its *density* as $\frac{|\mathcal{C}|}{q^n}$. Naturally, the higher the density is, the higher the rate of the code is.

B. Linear Error Scrubbing Codes

In this section, we consider symmetric errors, and assume that the memory scrubs the codeword as soon as a new error appears. That is, the error set

$$\varepsilon = \{(e_1, e_2, \cdots, e_n) \mid \sum_{i=1}^{n} |e_i| \leq 1,\ e_i \in \mathbb{Z} \text{ for all } i\}.$$

For simplicity, we also assume that $q \to \infty$. We will discuss the case of finite q later in the section.

We first present a new linear code construction of its general form, for $n \geq 4$. We will show later how to set its parameters to achieve optimal performance. Note that given two vectors $\vec{a} = (a_1, a_2, \cdots, a_n)$ and $\vec{b} = (b_1, b_2, \cdots, b_n)$, $\vec{a} \cdot \vec{b} = \sum_{i=1}^{n} a_i b_i$. Given an integer i, $i \cdot \vec{a} = (ia_1, ia_2, \cdots, ia_n)$.

Construction 2. Linear Error-Scrubbing Code. *Build a t-error-scrubbing code $\mathcal{C}$ for $n \geq 4$ cells as follows. Let $t \geq 1$ be an integer. Let $\vec{a} = (a_1, a_2, \cdots, a_n)$ and $\vec{b} = (b_1, b_2, \cdots, b_n)$ be two vectors, where a_i, b_i are positive integers for all i. Let*

$$V = \max_{\vec{e} \in \varepsilon} \vec{e} \cdot \vec{b} - \min_{\vec{e} \in \varepsilon} \vec{e} \cdot \vec{b} + 1 = 1 + 2 \max_i b_i.$$

For $i = 0, 1, \cdots, \lfloor \frac{\vec{a} \cdot \vec{b}}{V} \rfloor - 1$, let $\mathcal{C}_i = \{(c_1, c_2, \cdots, c_n) \mid \sum_{j=1}^{n} b_j c_j \equiv iV \mod (t \cdot \vec{a} \cdot \vec{b})\}$. Let $\mathcal{C} = \bigcup_{i=0}^{\lfloor \frac{\vec{a} \cdot \vec{b}}{V} \rfloor - 1} \mathcal{C}_i$.

The scrubbing function *$f : \mathcal{S}_{n,\infty} \to \mathcal{S}_{n,\infty}$ is defined as follows. Let $\vec{s} \in \mathcal{S}_{n,\infty}$ be any cell state. If there exists a codeword $\vec{c} \in \mathcal{C}$, an integer $i \in \{0, 1, \cdots, t-1\}$ and an error $\vec{e} \in \varepsilon$ such that $\vec{s} = \vec{c} + i\vec{a} + \vec{e}$ and $\vec{c} + i\vec{a} > \vec{s}$, then $f(\vec{s}) = \vec{c} + i\vec{a}$. If there exists a codeword $\vec{c} \in \mathcal{C}$, an integer $i \in \{0, 1, \cdots, t-2\}$ and an error $\vec{e} \in \varepsilon$ such that $\vec{s} = \vec{c} + i\vec{a} + \vec{e}$ and $\vec{s} > \vec{c} + i\vec{a}$, then $f(\vec{s}) = \vec{c} + (i+1)\vec{a}$. If $\vec{s}$ belongs to neither of the above two cases, then $f(\vec{s}) = \vec{s}$.*

Lemma 3. *Let $\mathcal{C}$ be the code of Construction 2. Let $\vec{c} \in \mathcal{C}$ be a codeword. Then, the*

decoding sphere of $\vec{c}$ *is* $B_{\vec{c}} = \{\vec{c} + i \cdot \vec{a} + \vec{e} \mid i \in \{0, 1, \cdots, t-1\};\ \vec{e} \in \varepsilon\}$.

The following theorem shows that the code $\mathcal{C}$ of Construction 2 is a t-error-scrubbing code.

Theorem 4. *Let* $\mathcal{C}$ *be the code of Construction 2. Then, for any two codewords* $\vec{c}_1$ *and* $\vec{c}_2$ *of* $\mathcal{C}$*, we have* $B_{\vec{c}_1} \cap B_{\vec{c}_2} = \emptyset$.

Proof. For any cell state $\vec{s} \in \mathcal{S}_{n,\infty}$, let us define its *signature* as $sig(\vec{s}) = (\vec{b} \cdot \vec{s} \bmod t \cdot (\vec{a} \cdot \vec{b}))$. For any codeword $\vec{c} \in \mathcal{C}$, let $G(\vec{c})$ denote the set of signatures of the cell states in the decoding sphere of $\vec{c}$. That is, $G(\vec{c}) = \{sig(\vec{s}) \mid \vec{s} \in B_{\vec{c}}\}$. Let $b_{max} = \max_i b_i$. Construction 2 shows that $\mathcal{C} = \bigcup_{i=0,1,\cdots,\lfloor \frac{\vec{a}\cdot\vec{b}}{V} \rfloor - 1} \mathcal{C}_i$, where each $\mathcal{C}_i$ contains a set of codewords. Then, it is simple to verify that if $\vec{c} \in \mathcal{C}_i$, then $G(\vec{c}) \subseteq \{iV + j \cdot \vec{a} \cdot \vec{b} + k \mod t \cdot \vec{a} \cdot \vec{b} \mid j \in \{0, 1, \cdots, t-1\};\ k \in \{-b_{max}, -b_{max}+1, \cdots, b_{max}\}\}$.

Let $\vec{c}_1$ and $\vec{c}_2$ be two different codewords in $\mathcal{C}$. If $\vec{c}_1 \in \mathcal{C}_i$ and $\vec{c}_2 \in \mathcal{C}_j$ for some $i \neq j$, then it is simple to see that $G(\vec{c}_1) \cap G(\vec{c}_2) = \emptyset$, so $B_{\vec{c}_1} \cap B_{\vec{c}_2} = \emptyset$. Now suppose that $\vec{c}_1 \in \mathcal{C}_i$ and $\vec{c}_2 \in \mathcal{C}_i$ for some i. We will prove $B_{\vec{c}_1} \cap B_{\vec{c}_2} = \emptyset$ by contradiction. Assume there exists a cell state $\vec{s} \in B_{\vec{c}_1} \cap B_{\vec{c}_2}$. It is simple to verify that for any codeword $\vec{c} \in \mathcal{C}$ and any cell state $\vec{s}_0 \in B_{\vec{c}}$, $sig(\vec{s}_0) - sig(\vec{c}) \bmod t \cdot \vec{a} \cdot \vec{b}$ is a function of $\vec{s}_0 - \vec{c}$, and no two cell states in $B_{\vec{c}}$ have the same signature. Since $sig(\vec{c}_1) = iV = sig(\vec{c}_2)$, we get $\vec{s} - \vec{c}_1 = \vec{s} - \vec{c}_2$. So $\vec{c}_1 = \vec{c}_2$, which is not true. So $B_{\vec{c}_1} \cap B_{\vec{c}_2} = \emptyset$. □

We now present a specific code construction.

Construction 5. *Let* $\vec{a} = (1, 1, \cdots, 1)$ *and* $\vec{b} = (1, 2, \cdots, n)$*. Then, use Construction 2 to build a* t*-error-scrubbing code* $\mathcal{C}$.

Theorem 6. *Let* $\mathcal{C}$ *be the* t*-error-scrubbing code of Construction 5. Its density is*

$$\frac{2\lfloor \frac{n(n+1)}{2(2n+1)} \rfloor}{tn(n+1)} = \Theta(\frac{1}{tn}),$$

which is asymptotically optimal.

Proof. Let V and C_i be as defined in Construction 2. Then for the code $\mathcal{C}$ of Construction 5, $V = 2n + 1$. A cell state $(c_1, c_2, \cdots, c_n)$ is in C_i if and only if $\sum_{j=1}^{n} jc_j \equiv i(2n+1) \bmod \frac{tn(n+1)}{2}$. Whatever values $c_2, c_3, \cdots, c_n$ take, the above equation produces $c_1 \equiv i(2n+1) - \sum_{j=2}^{n} jc_j \bmod \frac{tn(n+1)}{2}$. So $\lim_{q\to\infty} \frac{|\mathcal{C}_i|}{q^n} = \frac{2}{tn(n+1)}$. So the density of $\mathcal{C}$ is $\lim_{q\to\infty} \frac{|\mathcal{C}|}{q^n} = \lim_{q\to\infty} \frac{|\bigcup_{i=0}^{\lfloor \frac{n(n+1)}{2}/(2n+1) \rfloor - 1} \mathcal{C}_i|}{q^n} = \lfloor \frac{n(n+1)}{2(2n+1)} \rfloor \cdot \frac{2}{tn(n+1)} = \Theta(\frac{1}{tn})$.

To prove that the density of $\mathcal{C}$ is asymptotically optimal (up to a constant ratio), it is sufficient to show that for any t-error-scrubbing code $\mathcal{C}'$, $|B_{\vec{c}}| = \Omega(tn)$ for any codeword $\vec{c} \in \mathcal{C}'$. For $i = 1, 2, \cdots, n$, let $\vec{e}_i \in \varepsilon$ be the vector where the i-th element is 1 and all other elements are 0. Let $\vec{s}_0 = \vec{c}$; for $i = 1, 2, \cdots, t-1$, let $\vec{s}_i = f(\vec{s}_{i-1} + \vec{e}_1)$. For $i = 0, 1, \cdots, t-1$, let $S_i = \{\vec{s}_i + \vec{e}_j \mid 1 \le j \le n\}$. It is simple to see that $\forall\ i$, $S_i \subseteq B_{\vec{c}}$; also, $\forall\ i \neq j$, $S_i \cap S_j = \emptyset$ (because the cell states in S_i and S_j have different values in terms of the summation of cell levels). So $|B_{\vec{c}}| \ge \sum_{i=0}^{t-1} |S_i| = tn = \Omega(tn)$. □

We can choose different values for the vector $\vec{a} = (a_1, a_2, \cdots, a_n)$ to further increase the code's density. The density shown in the above theorem is upper bounded by $1/t(2n+1)$. The following theorem shows that the code density can reach this value through a different set of parameters. The tradeoff is to increase some cell levels by more than one in a scrubbing operation. We skip the proof of the following theorem due to its similarity to the previous analysis.

Theorem 7. *Let W be the smallest integer such that $W \ge \frac{n(n+1)}{2}$ and W is a multiple of $2n+1$. There exists an integer vector $\vec{a} = (a_1, a_2, \cdots, a_n)$ such that $\sum_{i=1}^{n} ia_i = W$ and $\forall\ i$, $a_i = 1$ or 2. Let $\vec{b} = (1, 2, \cdots, n)$. With the above parameter vectors $\vec{a}$ and $\vec{b}$, we can use Construction 2 to build a t-error-scrubbing code $\mathcal{C}$. The density of $\mathcal{C}$ is $\frac{1}{t(2n+1)}$.*

The above codes are for $n \geq 4$. When $n = 1, 2, 3$, we can build t-error-scrubbing codes of density $\frac{1}{t+2}$, $\frac{1}{3t+2}$ and $\frac{1}{7t}$, respectively. We summarize them with the following theorem.

Theorem 8. *When $n = 1, 2, 3$, there exist t-error-scrubbing codes of density $\frac{1}{t+2}$, $\frac{1}{3t+2}$ and $\frac{1}{7t}$, respectively.*

Proof. The proof is constructive. We skip the analysis for $n = 1$ due to its simplicity. (Its codewords are cell levels that are multiples of $t+2$.) When $n = 2$, let the code $\mathcal{C} = \{(c_1, c_2) \mid c_1 + 2c_2 \equiv 0 \mod (3t+2)\}$. When $n = 3$, let the code $\mathcal{C} = \{(c_1, c_2, c_3) \mid c_1 + 2c_2 + 4c_3 \equiv 0 \mod 7t\}$. For both codes, given a codeword (c_1, c_2) or (c_1, c_2, c_3) in $\mathcal{C}$, for $i = 0, 1, \cdots, t-1$, call the cell state $(c_1 + i, c_2 + i)$ or $(c_1 + i, c_2 + i, c_3 + i)$ the "i-shift" of the codeword. The decoding sphere of every codeword consists of the cell states within L_1 distance one from one of the t shifts of the codeword. The scrubbing function $f : \mathcal{S}_{n,\infty} \to \mathcal{S}_{n,\infty}$ is defined as follows. If a cell state $\vec{s}$ is at L_1 distance one from the i-shift of a codeword for some $i \in \{0, 1, \cdots, t-1\}$ and that i-shift is above $\vec{s}$, then $f(\vec{s})$ equals the i-shift of that codeword. If $\vec{s}$ is at L_1 distance one from the i-shift of a codeword for some $i \in \{0, 1, \cdots, t-2\}$ and $\vec{s}$ is above that i-shift, then $f(\vec{s})$ equals the $(i+1)$-shift of that codeword. If neither of the above two cases is true, then $f(\vec{s}) = \vec{s}$. It is not difficult to verify that the decoding spheres of codewords do not intersect. In fact, for both codes, the decoding spheres form a perfect packing in the L_1-metric space of dimension n. It is not difficult to see that the densities of the two codes are $1/(3t+2)$ and $1/7t$, respectively. □

Construction 2 can be generalized to correct other types of errors, including asymmetric errors, errors of large L_1-metric sizes, etc., by adjusting the parameters.

The t-error-scrubbing codes can correct errors whose total size (in the L_1 metric) is up to t. Let us compare them to conventional error-correcting codes that can correct

errors of the same size. For a conventional code, the decoding sphere for a codeword $(c_1, c_2, \cdots, c_n)$ is $B = \{(s_1, s_2, \cdots, s_n) \mid \sum_{i=1}^{n} |s_i - c_i| \leq t\}$. Since $|B| = \Omega(t^n)$, by the sphere-packing bound, the density of a conventional error-correcting code is $O(\frac{1}{t^n})$. The density of a t-error-scrubbing code, $\Theta(\frac{1}{tn})$, can be substantially better.

When q is finite, some codewords may not be able to scrub t errors because their cell levels are too close to the maximum value $q-1$. In practice, the memory can read cells to see how close they are to the limit of decodability, and adaptively schedule block erasures for refreshing codewords.

C. Modular Error Scrubbing Codes

In flash memories, errors in cell levels can be caused by several mechanisms, including read disturbs, write disturbs, charge leakage, etc. [4] They often change the cell levels in one direction more significantly than the other. In this section, we consider a more general form of errors, where at most $x \leq n$ cell levels can have errors between two scrubbing operations, and the errors make every cell level decrease by at most d_{min} and increase by at most d_{max}. That is, the error set is $\varepsilon = \{(e_1, e_2, \cdots, e_n) \mid e_i \in \{-d_{min}, -d_{min}+1, \cdots, d_{max}\}$ for all i; $|\{i|e_i \neq 0\}| \leq x\}$. The error set studied in the previous section is a special case with $x = 1$ and $d_{min} = d_{max} = 1$. For simplicity, we assume $q \to \infty$. The case of finite q can be processed the same way as before.

We first present a t-error-scrubbing code construction of its general form based on the modular technique. The modular technique has been proposed in [5], where strong codes for correcting asymmetric limited-magnitude errors have been presented. In this section, we use the modular technique for error-scrubbing coding.

Construction 9. Modular Error-Scrubbing Code.

Build a t-error-scrubbing code $\mathcal{C}$ as follows. Let $\ell = d_{min} + d_{max} + 1$. Let $\mathcal{D} \subseteq$

$\{0,1,\cdots,\ell-1\}^n$ be an error-correcting code of alphabet $\{0,1,\cdots,\ell-1\}$ and length n, which can correct x errors. (For code $\mathcal{D}$, an error is the distortion of one of its n symbols.) Then, a cell state $\vec{c}$ is a codeword in $\mathcal{C}$ if and only if there exists $\vec{s}\in\mathcal{D}$ and non-negative integers $a_1,a_2,\cdots,a_n$ and b such that $\min\{a_1,a_2,\cdots,a_n\}=0$ and $\vec{c}=\vec{s}+\ell\cdot(a_1,a_2,\cdots,a_n)+b\cdot(t\ell,t\ell,\cdots,t\ell)$.

The scrubbing function $f:\mathcal{S}_{n,\infty}\to\mathcal{S}_{n,\infty}$ is defined as follows. Let

$$\vec{s}=(s_1,s_2,\cdots,s_n)\in S_{n,\infty}$$

be any cell state. If there exists a codeword $\vec{c}\in\mathcal{C}$, an integer $i\in\{0,1,\cdots,t-1\}$ and an error $\vec{e}\in\varepsilon$ such that $\vec{s}=\vec{c}+i\cdot(\ell,\ell,\cdots,\ell)+\vec{e}$ and $\vec{c}+i\cdot(\ell,\ell,\cdots,\ell)>\vec{s}$, then $f(\vec{s})=\vec{c}+i\cdot(\ell,\ell,\cdots,\ell)$. If there exists a codeword $\vec{c}=(c_1,c_2,\cdots,c_n)\in\mathcal{C}$, an integer $i\in\{0,1,\cdots,t-2\}$ and an error $\vec{e}\in\varepsilon$ such that $\vec{s}=\vec{c}+i\cdot(\ell,\ell,\cdots,\ell)+\vec{e}$ and $s_j>c_j+i\ell$ for some $j\in\{1,2,\cdots,n\}$, then $f(\vec{s})=\vec{c}+(i+1)\cdot(\ell,\ell,\cdots,\ell)$. If $\vec{s}$ belongs to neither of the above two case, then $f(\vec{s})=\vec{s}$.

We now show that the code built by Construction 9 is a t-error-scrubbing code.

Theorem 10. *Let $\mathcal{C}$ be the code of Construction 9. Then, for any two codewords $\vec{c}_1$ and $\vec{c}_2$ of $\mathcal{C}$, we have $B_{\vec{c}_1}\cap B_{\vec{c}_2}=\emptyset$.*

Proof. Let $\vec{c}_1=\vec{s}_1+\ell\cdot(a_1,a_2,\cdots,a_n)+b\cdot(t\ell,t\ell,\cdots,t\ell)$ and $\vec{c}_2=\vec{s}_2+\ell\cdot(a'_1,a'_2,\cdots,a'_n)+b'\cdot(t\ell,t\ell,\cdots,t\ell)$. Here $\vec{s}_1,\vec{s}_2,a_i,a'_i,b,b'$ are defined the way as in Construction 9. Without loss of generality (WLOG), assume that $a_{j_1}=a'_{j_2}=0$. For every cell state $\vec{v}=(v_1,v_2,\cdots,v_n)$, define its "*signature*" as $sig(\vec{v})=(v_1 \mod \ell, v_2 \mod \ell,\cdots,v_n \mod \ell)$. Then $sig(\vec{c}_1)=\vec{s}_1$ and $sig(\vec{c}_2)=\vec{s}_2$. To prove $B_{\vec{c}_1}\cap B_{\vec{c}_2}=\emptyset$, consider three cases.

- Case one: $sig(\vec{c}_1)\neq sig(\vec{c}_2)$. Let $\vec{v}$ be a cell state in $B_{\vec{c}_1}$. Then, $\vec{v}=\vec{c_1}+i\cdot(\ell,\ell,\cdots,\ell)+\vec{e}$ for some $i\in\{0,1,\cdots,t-1\}$ and $\vec{e}\in\varepsilon$. So $sig(\vec{v})=sig(\vec{s_1}+\vec{e})$

is a vector in $\{0, 1, \cdots, \ell - 1\}^n$ that is within Hamming distance x from $\vec{s}_1$. Since $\mathcal{D}$ is an error-correcting code that corrects any x errors, $sig(\vec{v})$ cannot be any vector within Hamming distance x from $\vec{s}_2$. So $\vec{v} \notin B_{\vec{c}_2}$. So $B_{\vec{c}_1} \cap B_{\vec{c}_2} = \emptyset$.

- Case two: $sig(\vec{c}_1) = sig(\vec{c}_2)$, and $b \neq b'$. WLOG, assume that $b < b'$. In this case, let $\vec{s}_1 = \vec{s}_2 = (s_1, s_2, \cdots, s_n)$. Then for any cell state in $B_{\vec{c}_1}$, its j_1-th element is at most $s_{j_1} + bt\ell + (t-1)\ell + d_{max} < s_{j_1} + b't\ell - d_{min}$, while the j_1-th element of a cell state in $B_{\vec{c}_2}$ is at least $s_{j_1} + a'_{j_1}\ell + b't\ell - d_{min}$. So $B_{\vec{c}_1} \cap B_{\vec{c}_2} = \emptyset$.

- Case three: $sig(\vec{c}_1) = sig(\vec{c}_2)$, $b = b'$, and there exists some j_3 such that $a_{j_3} \neq a'_{j_3}$. WLOG, assume that $a_{j_3} < a'_{j_3}$. In this case, let $\vec{s}_1 = \vec{s}_2 = (s_1, s_2, \cdots, s_n)$. Let $(d_1, d_2, \cdots, d_n) = \vec{s}_1 + \ell \cdot (a_1, a_2, \cdots, a_n) + b \cdot (t\ell, t\ell, \cdots, t\ell) + \alpha \cdot (\ell, \ell, \cdots, \ell) + \vec{e}_1$ be a cell state in $B_{\vec{c}_1}$, and let $(d'_1, d'_2, \cdots, d'_n) = \vec{s}_1 + \ell \cdot (a'_1, a'_2, \cdots, a'_n) + b \cdot (t\ell, t\ell, \cdots, t\ell) + \alpha' \cdot (\ell, \ell, \cdots, \ell) + \vec{e}_2$ be a cell state in $B_{\vec{c}_2}$. Here $\alpha, \alpha' \in \{0, 1, \cdots, t-1\}$, and $\vec{e}_1 = (\beta_1, \beta_2, \cdots, \beta_n)$, $\vec{e}_2 = (\beta'_1, \beta'_2, \cdots, \beta'_n)$ are two errors in ε. Consider two subcases.

  - Subcase one: $a_{j_3} + \alpha \neq a'_{j_3} + \alpha'$. WLOG, assume that $a_{j_3} + \alpha < a'_{j_3} + \alpha'$. Then, $d_{j_3} = s_{j_3} + bt\ell + (a_{j_3} + \alpha)\ell + \beta_{j_3} \leq s_{j_3} + bt\ell + (a'_{j_3} + \alpha' - 1)\ell + d_{max} < s_{j_3} + bt\ell + (a'_{j_3} + \alpha')\ell - d_{min} \leq d'_{j_3}$.

  - Subcase two: $a_{j_3} + \alpha = a'_{j_3} + \alpha'$. Since $a_{j_3} < a'_{j_3}$, we get $\alpha > \alpha'$. Then, $d_{j_2} = s_{j_2} + bt\ell + (a_{j_2} + \alpha)\ell + \beta_{j_2} \geq s_{j_2} + bt\ell + (\alpha' + 1)\ell - d_{min} > s_{j_2} + bt\ell + \alpha'\ell + d_{max} \geq d'_{j_2}$.

So in both subcases, $(d_1, d_2, \cdots, d_n) \neq (d'_1, d'_2, \cdots, d'_n)$. Therefore, $B_{\vec{c}_1} \cap B_{\vec{c}_2} = \emptyset$. The theorem is proved. □

Theorem 11. *Let $\mathcal{C}$ be the t-error-scrubbing code of Construction 9. Let $\mathcal{D}$ be the x-error-correcting code used in Construction 9. Let $\ell = d_{min} + d_{max} + 1$. Then, the*

density of $\mathcal{C}$ *is* $\frac{|\mathcal{D}|}{t\ell^n}$.

Proof. For every cell state $\vec{v} = (v_1, v_2, \cdots, v_n)$, define its "*signature*" as $sig(\vec{v}) = (v_1 \mod \ell, v_2 \mod \ell, \cdots, v_n \mod \ell)$. For every codeword $\vec{c} \in \mathcal{C}$, we call $\vec{c} + i \cdot (\ell, \ell, \cdots, \ell)$ the i-shift of $\vec{c}$. Then the 0-shift, 1-shift, $\cdots$, $(t-1)$-shift of $\vec{c}$ are all in $B_{\vec{c}}$. Let $\vec{d} = (d_1, d_2, \cdots, d_n)$ be a codeword of $\mathcal{D}$. The density of cell states of signature $\vec{d}$ – defined as the number of such cell states divided by q^n – is $1/\ell^n$. We now show that every cell state of signature $\vec{d}$ must be the i-shift of a codeword of $\mathcal{C}$ for some $i \in \{0, 1, \cdots, t-1\}$.

Let $\vec{s} = (d_1 + k_1\ell, d_2 + k_2\ell, \cdots, d_n + k_n\ell)$ be a general cell state of signature $\vec{d}$. Let j be the integer in $\{1, 2, \cdots, n\}$ such that $k_j = \min\{k_1, k_2, \cdots, k_n\}$. So $\vec{s} = (d_1 + (k_1 - k_j)\ell + k_j\ell, d_2 + (k_2 - k_j)\ell + k_j\ell, \cdots, d_n + (k_n - k_j)\ell + k_j\ell) = (d_1 + (k_1 - k_j)\ell + \lfloor \frac{k_j}{t} \rfloor t\ell + (k_j \mod t)\ell, d_2 + (k_2 - k_j)\ell + \lfloor \frac{k_j}{t} \rfloor t\ell + (k_j \mod t)\ell, \cdots, d_n + (k_n - k_j)\ell + \lfloor \frac{k_j}{t} \rfloor t\ell + (k_j \mod t)\ell)$. So $\vec{s}$ is the $(k_j \mod t)$-shift of the codeword $(d_1 + (k_1 - k_j)\ell + \lfloor \frac{k_j}{t} \rfloor t\ell, d_2 + (k_2 - k_j)\ell + \lfloor \frac{k_j}{t} \rfloor t\ell, \cdots, d_n + (k_n - k_j)\ell + \lfloor \frac{k_j}{t} \rfloor t\ell) \in \mathcal{C}$.

Since every codeword of $\mathcal{C}$ has exactly t shifts in its decoding sphere, the density of $\mathcal{C}$ is $|\mathcal{D}|/t\ell^n$. □

Let $\ell = d_{min} + d_{max} + 1$. By the sphere-packing bound, the density of a conventional error-correcting code that can correct t errors in the error set ε is $O(\frac{1}{n^{tx}(\ell-1)^{tx}})$. So when $\frac{|\mathcal{D}|}{\ell^n} = \Omega(\frac{1}{n^x(\ell-1)^x})$, the t-error-scrubbing code can significantly outperform the conventional code. For example, assume $n = 2^m - 1$, $d_{max} = 1$, $d_{min} = 0$, $x = 1$ and $\mathcal{D}$ is the $(2^m - 1, 2^m - m - 1)$ Hamming code. Then the density of the t-error-scrubbing code is $\frac{|\mathcal{D}|}{t\ell^n} = \frac{1}{t \cdot 2^m} = \frac{1}{t(n+1)}$, which is asymptotically optimal in t and can be much higher than the density of a conventional code, $O(\frac{1}{n^t})$.

CHAPTER III

OPTIMIZED CELL PROGRAMMING FOR FLASH MEMORIES

Flash memory cells are a unique storage medium in the sense that when they are programmed with multiple rounds of charge injection, their charge levels can only monotonically increase. It is interesting to study how to program the cells accurately, as the precision of cell programming determines the storage capacity of flash memories [11]. In this chapter, we focus on the cell programming strategy that optimizes the expected performance. The performance criteria considered here include two metrics that are suitable for the multi-level cell technology and the rank modulation technology, respectively. Knowing how well cells can be programmed on average is useful for studying the storage capacity of cell ensembles. We present an effective algorithm for finding the optimal programming strategy, which can in turn be used to program cells efficiently.

A. The Cell Programming Problem

Without loss of generality (w.l.o.g.), we assume that initially, the cell level is 0. A cell can be programmed using at most t rounds of charge injection. The objective is to make the final cell level be close to a target value $\theta \in [0, \mathcal{L}]$, where $\mathcal{L}$ is an upper bound determined by the physics of flash memories. There is a cost $C(x)$ associated with the final cell level x, and the function $C(x)$ monotonically increases with $|x - \theta|$. Two forms of $C(x)$ will be introduced later.

We assume that in each round of charge injection, the flash memory can choose the aimed increment of the cell level to be $i\Delta$ for some $i \in \{0, 1, 2, 3, \cdots\}$. Here Δ models the minimum resolution of the programming circuit. Let $\epsilon \in (0, 1)$ and $\delta > 0$

be two parameters. To model the noisy charge-injection process, we assume that if the aimed increment of the cell level is $i\Delta$, the actual increment of the cell level is randomly distributed in the range $[i\Delta(1-\epsilon), i\Delta(1+\delta))$.[1] For simplicity, we assume the distribution is a uniform random distribution. More practical noise models can be studied in the future.

In this section, we consider two families of cost functions.

Definition 12.. (Cost function $C(x)$ for MLC and Rank Modulation) *In the multi-level cell (MLC) technology, the final cell level should be close to one of a set of discrete levels. It is appropriate to define $C(x)$ as*

$$C(x) = |x - \theta|^p$$

for some positive integer p.

In the rank modulation technology [13, 14, 15], the objective is usually to shift the cell level above *a certain value θ. It is appropriate to define $C(x)$ as*

$$C(x) = \begin{cases} \infty & , \text{ if } x < \theta \\ (x-\theta)^p & , \text{ if } x \geq \theta \end{cases}$$

for some positive integer p.

Let $i_1\Delta, i_2\Delta, \cdots, i_t\Delta$ denote the aimed increment of the cell level in the t rounds of charge injection, and let $x_1, x_2, \cdots, x_t$ denote the the actual cell level after each round. After the j-th round, the flash memory can measure x_j and adaptively choose the aimed level increment $i_{j+1}\Delta$ for the next round. The objective of the cell programming problem is to find the adaptive strategy of selecting $i_1, i_2, \cdots, i_t$ such that

[1]The inclusion and exclusion of the boundary values are chosen for mathematical convenience, and are easy to deal with in practice.

the expected cost of the final cell level, $E(C(x_t))$, is minimized. (Here $E(x)$ is the expectation of the random variable x.)

B. Adaptive Cell Programming

Given that the cell level is x_j after $j < t$ rounds of charge injection, how to choose the aimed level increment $i_{j+1}\Delta$ of the next round? We define two functions, $\mathcal{A}(x; i)$ and $\alpha(x; i; j)$, for the computation.

Definition 13.. (FUNCTIONS $\mathcal{A}(x; i)$ AND $\alpha(x; i; j)$)

$\mathcal{A}(x; i)$ is the minimum achievable expected cost of the final cell level given that (1) the current cell level is $\theta + x$, and (2) we can program the cell with i more rounds of charge injection.

$\alpha(x; i; j)$ is the minimum achievable expected cost of the final cell level given that (1) the current cell level is $\theta + x$, (2) we can program the cell with i more rounds of charge injection, and (3) in the first round of the i rounds of charge injection, we choose the aimed level increment to be $j\Delta$.

It is simple to see that $\mathcal{A}(x; i) = \min_{j=0,1,2\cdots} \alpha(x; i; j)$. For the cell programming problem, since the initial cell level is $0 = \theta + (-\theta)$ and t rounds of charge injection can be used, the objective is to find a strategy that makes the final cell level's expected cost be $\mathcal{A}(-\theta; t)$. During the programming process, given that the cell level is x_j after $j < t$ rounds of charge injection, the flash memory should adaptively choose $i_{j+1}\Delta$ as the aimed level increment of the $(j + 1)$-th round such that i_{j+1} minimizes the value of $\alpha(x_j - \theta; t - j; i_{j+1})$.

The cost function we consider is for MLC or rank modulation, which is shown in Definition 12. Let us compute some initial values of $\mathcal{A}(x; i)$ – particularly, $\mathcal{A}(x; 1)$ – for them.

1. When the Cost Function Is for MLC

The cost function for MLC is $C(x) = |x-\theta|^p$. For simplicity, we show how to compute $\mathcal{A}(x;1)$ when $p = 2$. The other values of p can be dealt with similarly.

When $p = 2$, we have

$$\begin{aligned}
& \mathcal{A}(x;1) \\
= \; & \min_{j=0,1,2\cdots} \; \alpha(x;1;j) \\
= \; & \min\{ \quad \alpha(x;1;0), \\
& \min_{j=1,2,3\cdots} \; \int_{\theta+x+j\Delta(1-\epsilon)}^{\theta+x+j\Delta(1+\delta)} \frac{1}{j\Delta(\epsilon+\delta)} \cdot |y-\theta|^2 dy\} \\
= \; & \min\{ \quad x^2, \\
& \min_{j=1,2,3\cdots} \; \frac{1}{j\Delta(\epsilon+\delta)} \int_{x+j\Delta(1-\epsilon)}^{x+j\Delta(1+\delta)} y^2 dy\} \\
= \; & \min_{j=0,1,2\cdots} \; x^2 + j\Delta(2+\delta-\epsilon)x + \tfrac{1}{3}j^2\Delta^2(3+ \\
& \qquad\qquad\qquad\quad 3\delta - 3\epsilon + \delta^2 - \delta\epsilon + \epsilon^2)
\end{aligned}$$

To see which value of j minimizes the above equation, define $f(j) = x^2 + j\Delta(2+\delta-\epsilon)x + \frac{1}{3}j^2\Delta^2(3+3\delta-3\epsilon+\delta^2-\delta\epsilon+\epsilon^2)$. Since $0 < \epsilon < 1$ and $\delta > 0$, $3+3\delta-3\epsilon+\delta^2-\delta\epsilon+\epsilon^2 > 0$, so $f(j)$ is convex. By setting $\frac{df(j)}{dj} = 0$, we find that $f(j)$ is minimized when $j = \frac{-3x(2+\delta-\epsilon)}{2\Delta(3+3\delta-3\epsilon+\delta^2-\delta\epsilon+\epsilon^2)}$ (assuming that j does not have to be an integer). We can see that the above value for j is positive if and only if x is negative. Since j actually needs to be a non-negative integer, we find that to minimize $f(j)$, j should take the following value j^*:

$$j^* = \begin{cases} \lceil \frac{-3(2+\delta-\epsilon)x}{2\Delta(3+3\delta-3\epsilon+\delta^2-\delta\epsilon+\epsilon^2)} - 0.5 \rceil & \text{, if } x < 0 \\ 0 & \text{, if } x \geq 0 \end{cases}$$

Let $\gamma = \frac{2\Delta(3+3\delta-3\epsilon+\delta^2-\delta\epsilon+\epsilon^2)}{3(2+\delta-\epsilon)}$. Then when $x < 0$, $j^* = \lceil \frac{-x}{\gamma} - 0.5 \rceil$. So for $i =$

$1, 2, \cdots, \lceil \frac{\theta}{\gamma} - 1.5 \rceil$, when $x \in [-(i+0.5)\gamma, -(i-0.5)\gamma)$, we have $j^* = i$ and

$$\begin{aligned} \mathcal{A}(x;1) = \; & x^2 + i\Delta(2+\delta-\epsilon)x \\ & + \tfrac{1}{3} i^2 \Delta^2 (3 + 3\delta - 3\epsilon + \delta^2 - \delta\epsilon + \epsilon^2). \end{aligned}$$

Similarly, when $x \in [-0.5\gamma, 0)$, we have $j^* = 0$ and $\mathcal{A}(x;1) = x^2$. When $x \in [-\theta, -(\lceil \frac{\theta}{\gamma} - 0.5 \rceil - 0.5)\gamma)$, we have $j^* = \lceil \frac{\theta}{\gamma} - 0.5 \rceil$ and $\mathcal{A}(x;1) = x^2 + \lceil \frac{\theta}{\gamma} - 0.5 \rceil \Delta(2 + \delta - \epsilon)x + \frac{1}{3} \lceil \frac{\theta}{\gamma} - 0.5 \rceil^2 \Delta^2 (3 + 3\delta - 3\epsilon + \delta^2 - \delta\epsilon + \epsilon^2)$. When $x \geq 0$, we have $j^* = 0$ and $\mathcal{A}(x;1) = x^2$. Therefore, we can partition the domain of x, $[-\theta, \infty)$, into $\lceil \frac{\theta}{\gamma} + 0.5 \rceil$ regions, while in each region $\mathcal{A}(x;1)$ is a degree-2 polynomial. So $\mathcal{A}(x;1)$ is piecewise polynomial.

It is not hard to see that when $p \neq 2$, $\mathcal{A}(x;1)$ is also piecewise polynomial. For simplicity we skip the details.

2. When the Cost Function Is for Rank Modulation

The cost function for rank modulation is $C(x) = \infty$ if $x < \theta$ and $C(x) = (x-\theta)^p$ if $x \geq \theta$. For simplicity, we show how to compute $\mathcal{A}(x;1)$ when $p = 1$. The other values of p can be dealt with similarly.

We have $\mathcal{A}(x;1) = \min_{j=0,1,2\cdots} \alpha(x;1;j)$. The value of j that minimizes $\alpha(x;1;j)$ is the minimum integer that satisfies the constraint $\theta + x + j\Delta(1-\epsilon) \geq \theta$, which is $j = \lceil \frac{-x}{\Delta(1-\epsilon)} \rceil$. So if $x < 0$, we have

$$\begin{aligned} \mathcal{A}(x;1) \; & = \int_{\theta + x + \lceil \frac{-x}{\Delta(1-\epsilon)} \rceil \Delta(1-\epsilon)}^{\theta + x + \lceil \frac{-x}{\Delta(1-\epsilon)} \rceil \Delta(1+\delta)} \frac{1}{\lceil \frac{-x}{\Delta(1-\epsilon)} \rceil \Delta(\epsilon+\delta)} \cdot (y - \theta) dy \\ & = x + \lceil \tfrac{-x}{\Delta(1-\epsilon)} \rceil \Delta (1 + \tfrac{\delta - \epsilon}{2}) \end{aligned}$$

If $x \geq 0$, clearly $\mathcal{A}(x;1) = x$.

So for $i = 1, 2, \cdots, \lceil \frac{\theta}{\Delta(1-\epsilon)} \rceil - 1$, when $x \in [-i\Delta(1-\epsilon), -(i-1)\Delta(1-\epsilon))$, $\mathcal{A}(x;1) = x + i\Delta(1 + \frac{\delta - \epsilon}{2})$. When $x \in [-\theta, -(\lceil \frac{\theta}{\Delta(1-\epsilon)} \rceil - 1)\Delta(1-\epsilon))$, $\mathcal{A}(x;1) =$

$x + \lceil \frac{\theta}{\Delta(1-\epsilon)} \rceil \Delta(1 + \frac{\delta-\epsilon}{2})$. When $x \geq 0$, $\mathcal{A}(x;1) = x$. So we can partition the domain of x, $[-\theta, \infty)$, into $\lceil \frac{\theta}{\Delta(1-\epsilon)} \rceil + 1$ regions, while in each region, $\mathcal{A}(x;1)$ is a linear function. So $\mathcal{A}(x;1)$ is piecewise polynomial.

It is not hard to see that when $p \neq 1$, $\mathcal{A}(x;1)$ is also piecewise polynomial. For simplicity we skip the details.

C. Computing $\mathcal{A}(x;i)$ and $\alpha(x;i;j)$

When $i \geq 2$, we have

$$\mathcal{A}(x;i) = \min_{j=0,1,2\cdots} \alpha(x;i;j)$$

and

$$\alpha(x;i;j) = \int_{x+j\Delta(1-\epsilon)}^{x+j\Delta(1+\delta)} \frac{\mathcal{A}(y;i-1)}{j\Delta(\delta+\epsilon)} dy$$

for $j \geq 1$. (We have $\alpha(x;i;0) = \mathcal{A}(x;i-1)$.)

It will be interesting to find an effective approach to compute the general functions $\mathcal{A}(x;i)$ and $\alpha(x;i;j)$ using the above recursion. In this paper, we present an efficient algorithm using the property that they are both piecewise polynomials. (Note that this property of being piecewise polynomial has been proved for $\mathcal{A}(x;1)$. It will be shown that it holds for $\mathcal{A}(x;i)$ and $\alpha(x;i;j)$ with $i \geq 2$, too.)

Let us define some notations. Given integers i, j, let $p_{i,j}$ and

$$b_{i,j}(-1)\ ,\ b_{i,j}(0)\ ,\ b_{i,j}(1)\ ,\ \cdots\ ,\ b_{i,j}(p_{i,j})$$

be the numbers with the following properties: (1) $b_{i,j}(-1) > b_{i,j}(0) > b_{i,j}(1) > b_{i,j}(2) > \cdots > b_{i,j}(p_{i,j})$; (2) $b_{i,j}(-1) = \infty$, $b_{i,j}(0) = 0$, $b_{i,j}(p_{i,j}) = -\theta$; (3) for $k = 0, 1, \cdots, p_{i,j}$, the function $\alpha(x;i;j)$ is a polynomial of x when $x \in [b_{i,j}(k), b_{i,j}(k-1))$.

Given an integer $i \geq 1$, let q_i and

$$B_i(-1)\ ,\ B_i(0)\ ,\ B_i(1)\ ,\ \cdots\ ,\ B_i(q_i)$$

be the numbers with the following properties: (1) $B_i(-1) > B_i(0) > B_i(1) > \cdots > B_i(q_i)$; (2) $B_i(-1) = \infty$, $B_i(0) = 0$, $B_i(q_i) = -\theta$; (3) for $k = 0, 1, \cdots, q_i$, the function $\mathcal{A}(x; i)$ is a polynomial of x when $x \in [B_i(k), B_i(k-1))$.

1. Computing $\alpha(x; i; j)$ with $i \geq 2$

We first show how to compute $\alpha(x; i; j)$ with $i \geq 2$.

Given a real number $x \in [-\theta, \infty)$ and an integer $i \geq 1$, we call the unique integer $j \in \{0, 1, \cdots, q_i\}$ such that

$$x \in [B_i(j)\ ,\ B_i(j-1))$$

the " (i)-*index of* x", and denote it by

$$index(i; x).$$

Note that $index(i; x)$ decreases as x increases. Let us use $\mathcal{I}(i; x; j)$ to denote the set of (i)-indices of the real numbers in the interval

$$[x + j\Delta(1-\epsilon)\ ,\ x + j\Delta(1+\delta)).$$

We get

$$\begin{aligned}\mathcal{I}(i; x; j) = \{\ & index(i; x + j\Delta(1-\epsilon));\\ & index(i; x + j\Delta(1-\epsilon)) - 1;\\ & index(i; x + j\Delta(1-\epsilon)) - 2;\\ & \cdots\\ & \lim_{\nu \to 0^+} index(i; x + j\Delta(1+\delta) - \nu)\ \ \}\end{aligned}$$

The last element in the above set is a limit, because the interval $[x + j\Delta(1-\epsilon)\ , x + j\Delta(1+\delta))$ does not contain the boundary value $x + j\Delta(1+\delta)$.

Let us define the set $S_{i,j}$ (for $i \geq 2$) as

$$\begin{aligned} S_{i,j} \quad = \{ \quad & s \in (-\theta, 0)| \text{ either } s = B_{i-1}(k) - j\Delta(1-\epsilon) \\ & \text{for some } 0 \leq k \leq q_{i-1} - 1, \text{ or} \\ & s = B_{i-1}(k) - j\Delta(1+\delta) \\ & \text{for some } 0 \leq k \leq q_{i-1} - 1\}. \end{aligned}$$

Then we have the following lemma.

Lemma 14.. *We denote the $|S_{i,j}|$ numbers in the set $S_{i,j}$ by*

$$s_1, s_2, \cdots, s_{|S_{i,j}|}$$

such that $s_1 > s_2 > \cdots > s_{|S_{i,j}|}$. Also, let $s_0 = 0$ and $s_{|S_{i,j}|+1} = -\theta$. Then, for $k = 1, 2, \cdots, |S_{i,j}| + 1$, for any two numbers x_1, x_2 in the interval (s_k, s_{k-1}),

$$\mathcal{I}(i-1; x_1; j) = \mathcal{I}(i-1; x_2; j).$$

Proof. W.l.o.g., assume that $x_1 < x_2$. We just need to prove that (1)

$$index(i-1; x_1 + j\Delta(1-\epsilon)) = index(i-1; x_2 + j\Delta(1-\epsilon))$$

and (2)

$$\begin{aligned} & \lim_{\nu \to 0^+} index(i-1; x_1 + j\Delta(1+\delta) - \nu) \\ = \quad & \lim_{\nu \to 0^+} index(i-1; x_2 + j\Delta(1+\delta) - \nu). \end{aligned}$$

Let us prove condition (1) by contradiction. Assume that $index(i-1; x_1 + j\Delta(1-$

$\epsilon)) \neq index(i-1; x_2 + j\Delta(1-\epsilon))$. Then there must be some $B_{i-1}(k')$ such that

$$x_1 + j\Delta(1-\epsilon) < B_{i-1}(k') \leq x_2 + j\Delta(1-\epsilon).$$

So

$$x_1 < B_{i-1}(k') - j\Delta(1-\epsilon) \leq x_2.$$

Since

$$B_{i-1}(k') - j\Delta(1-\epsilon) \in S_{i,j} = \{s_1, s_2, \cdots, s_{|S_{i,j}|}\},$$

x_1 and x_2 cannot be in the same interval (s_k, s_{k-1}). That is a contradiction. So $index(i-1; x_1 + j\Delta(1-\epsilon)) = index(i-1; x_2 + j\Delta(1-\epsilon))$.

Condition (2) can be proved similarly. For simplicity, we skip the details. □

Theorem 15.. *We denote the $|S_{i,j}|$ numbers in the set $S_{i,j}$ by*

$$s_1, s_2, \cdots, s_{|S_{i,j}|}$$

such that $s_1 > s_2 > \cdots > s_{|S_{i,j}|}$. Also, let $s_0 = 0$ and $s_{|S_{i,j}|+1} = -\theta$. Then, for $k = 1, 2, \cdots, |S_{i,j}| + 1$, the function $\alpha(x; i; j)$ is a polynomial of x for $x \in (s_k, s_{k-1})$. Furthermore, it can be computed as follows. Let

$$u = \lim_{\nu \to 0^+} index(i-1; s_k + j\Delta(1-\epsilon) + \nu),$$

and let

$$v = \lim_{\nu \to 0^+} index(i-1; s_{k-1} + j\Delta(1+\delta) - \nu).$$

Then,

$$\begin{aligned} \alpha(x; i; j) = \; & \int_{x+j\Delta(1-\epsilon)}^{B_{i-1}(u-1)} \frac{\mathcal{A}(y; i-1)}{j\Delta(\epsilon+\delta)}\, dy + \\ & \sum_{k=v+1}^{u-1} \int_{B_{i-1}(k)}^{B_{i-1}(k-1)} \frac{\mathcal{A}(y; i-1)}{j\Delta(\epsilon+\delta)}\, dy + \\ & \int_{B_{i-1}(v)}^{x+j\Delta(1+\delta)} \frac{\mathcal{A}(y; i-1)}{j\Delta(\epsilon+\delta)}\, dy \end{aligned}$$

Proof. We know that

$$\alpha(x;i;j) = \int_{x+j\Delta(1-\epsilon)}^{x+j\Delta(1+\delta)} \frac{\mathcal{A}(y;i-1)}{j\Delta(\epsilon+\delta)} dy.$$

Since $x \in (s_k, s_{k-1})$, by Lemma 14, $index(i-1; x+j\Delta(1-\epsilon)) = \lim_{\nu\to 0^+} index(i-1; s_k+j\Delta(1-\epsilon)+\nu) = u$ and $\lim_{\nu\to 0^+} index(i-1; x+j\Delta(1+\delta)-\nu) = \lim_{\nu\to 0^+} index(i-1; s_{k-1}+j\Delta(1+\delta)-\nu) = v$. So in the above integration, we can partition the domain for y into smaller intervals, in each of which the function $\mathcal{A}(y;i-1)$ is a polynomial of y. So the way to compute $\alpha(x;i;j)$ in this theorem is correct.

$\mathcal{A}(y;i-1)$ is a polynomial of y for $y \in [B_{i-1}(u), B_{i-1}(u-1)) \supseteq [x+j\Delta(1-\epsilon), B_{i-1}(u-1))$ and for $y \in [B_{i-1}(v), B_{i-1}(v-1)) \supseteq [B_{i-1}(v), x+j\Delta(1+\delta))$. Also note that the value of $\sum_{k=v+1}^{u-1} \int_{B_{i-1}(k)}^{B_{i-1}(k-1)} \frac{\mathcal{A}(y;i-1)}{j\Delta(\epsilon+\delta)}\, dy$ is independent of $x \in (s_k, s_{k-1})$. Since polynomials are closed under integration and summation, we get that $\alpha(x;i;j)$ is a polynomial of x for $x \in (s_k, s_{k-1})$. □

The above theorem shows that $\alpha(x;i;j)$ is an integration of $\mathcal{A}(x;i-1)$. It is easy to see that if $\mathcal{A}(x;i-1)$ is a piecewise polynomial of degree d, then $\alpha(x;i;j)$ is a piecewise polynomial of degree at most $d+1$. As we will see, $\mathcal{A}(x;i)$ is also a piecewise polynomial of degree at most $d+1$.

Corollary 16.. *We denote the $|S_{i,j}|$ numbers in the set $S_{i,j}$ by $s_1, s_2, \cdots, s_{|S_{i,j}|}$ such that $s_1 > s_2 > \cdots > s_{|S_{i,j}|}$. Also, let $s_{-1} = \infty$, $s_0 = 0$ and $s_{|S_{i,j}|+1} = -\theta$. Then, for $k = 0, 1, 2, \cdots, |S_{i,j}|+1$, the function $\alpha(x;i;j)$ is a polynomial of x for $x \in [s_k, s_{k-1})$.*

Proof. Since the integration of a finite function is a continuous function, we get $\alpha(s_k;i;j) = \lim_{\nu\to 0^+} \alpha(s_k+\nu;i;j)$. With Theorem 15, it is not hard to see that the conclusion holds. □

With the algorithm in Theorem 15, we can partition the domain of x, $[-\theta, \infty)$,

into the intervals

$$[-\theta, s_{|S_{i,j}|}), [s_{|S_{i,j}|}, s_{|S_{i,j}|-1}), \cdots, [s_1, 0), [0, \infty)$$

and compute the polynomial $\alpha(x; i; j)$ for each interval. To simplify the future computation, if the polynomials in adjacent intervals happen to be the same, we merge them into one interval.

2. Computing $\mathcal{A}(x; i)$ with $i \geq 2$

In Section B, we have shown how to compute $\mathcal{A}(x; 1)$. We now show how to compute $\mathcal{A}(x; i)$ for $i \geq 2$.

It is easy to see that when $j \geq \lceil \frac{-x}{\Delta(1-\epsilon)} \rceil$, $\alpha(x; i; j) \geq \alpha(x; i; \lceil \frac{-x}{\Delta(1-\epsilon)} \rceil)$ (because setting the aimed level increment too high only increases the expected cost of the final cell level). So when $i \geq 2$, we have

$$\mathcal{A}(x; i) = \min_{j=0}^{\lceil \frac{-x}{\Delta(1-\epsilon)} \rceil} \alpha(x; i; j) \tag{3.1}$$

We first use the algorithm in Theorem 15 to compute the functions

$$\alpha(x; i; 0) \ , \ \alpha(x; i; 1), \cdots, \alpha(x; i; \lceil \frac{\theta}{\Delta(1-\epsilon)} \rceil).$$

(Note that when $x \in [-\theta, 0)$, $\lceil \frac{-x}{\Delta(1-\epsilon)} \rceil \leq \lceil \frac{\theta}{\Delta(1-\epsilon)} \rceil$.) Let $S_{i,j}$ be as defined before. And denote the $|S_{i,j}|$ numbers in the set $S_{i,j}$ by

$$s_1^{i,j}, s_2^{i,j}, \cdots, s_{|S_{i,j}|}^{i,j}$$

such that $0 > s_1^{i,j} > s_2^{i,j} > \cdots > s_{|S_{i,j}|}^{i,j} > -\theta$. We know that $\alpha(x; i; j)$ is a polynomial of x for x in each of the following intervals

$$[-\theta, s_{|S_{i,j}|}^{i,j}) \ , \ [s_{|S_{i,j}|}^{i,j} \ , \ s_{|S_{i,j}|-1}^{i,j}) \ , \ \cdots \ , \ [s_1^{i,j}, 0) \ , \ [0, \infty)$$

Given the integer $i \geq 2$, let us define the set P as

$$P = \bigcup_{j=0}^{\lceil \frac{\theta}{\Delta(1-\epsilon)} \rceil} \{s_1^{i,j}, s_2^{i,j}, \cdots, s_{|S_{i,j}|}^{i,j}\}$$

Let us alternatively denote the elements in P by

$$p_1, p_2, \cdots, p_{|P|}$$

such that

$$p_1 > p_2 > \cdots > p_{|P|}.$$

Also let $p_{-1} = \infty$, $p_0 = 0$ and $p_{|P|+1} = -\theta$. We naturally have the following conclusion.

Lemma 17.. *For $k = 0, 1, 2, \cdots, |P| + 1$, the function $\alpha(x; i; j)$ is a polynomial of x for $x \in [p_k, p_{k-1})$. (Here $i \geq 2$ and $0 \leq j \leq \lceil \frac{\theta}{\Delta(1-\epsilon)} \rceil$.)*

With the above observation, we can easily compute the function $\mathcal{A}(x; i)$ for x in each interval $[p_k, p_{k-1})$, where $k = 0, 1, \cdots, |P| + 1$. That is because by Equation 3.1, $\mathcal{A}(x; i)$ is the minimum of at most $\lceil \frac{\theta}{\Delta(1-\epsilon)} \rceil + 1$ known polynomials. The method of computation should be clear, so we skip its details. The only thing to note is that if these polynomials intersect, the interval $[p_k, p_{k-1})$ may need to be partitioned into more smaller intervals, such that in each smaller interval, $\mathcal{A}(x; i)$ is still a polynomial of x.

As before, after the above computation, if the polynomials for $\mathcal{A}(x; i)$ in adjacent intervals happen to be the same, we merge them into one interval for a more succinct representation.

D. Optimal Cell Programming Strategy

In this section, we describe the cell-programming strategy that minimizes the expected cost of the final cell level. Recall that at most t rounds of charge injection can be used for programming a cell. We use the algorithm described before to compute the functions $\mathcal{A}(x;i)$ for $i = 1, 2, \cdots, t$, and compute the functions $\alpha(x;i;j)$ for $i = 1, 2, \cdots, t$ and $j = 0, 1, 2, \ldots, \lceil \frac{\theta}{\Delta(1-\epsilon)} \rceil$. These functions are then stored in the storage system, to be looked up during the actual cell-programming process.[2]

For $i = 1, 2, \cdots, t$, let x_i denote the actual cell level after the i-th round of charge injection. Let $x_0 = 0$ denote the initial cell level. The objective of cell programming is to minimize the expectation of $C(x_t)$. The optimal cell-programming strategy is as follows:

For $i = 0, 1, \cdots, t-1$, set the aimed level increment in the $(i+1)$-th round of charge injection to be $j^*\Delta$ such that

$$\alpha(x_i - \theta; t-i; j^*) = \mathcal{A}(x_i - \theta; t-i).$$

It should be noted that once the functions $\mathcal{A}(x;i)$ and $\alpha(x;i;j)$ are stored, it is very efficient to look them up for the actual programming of cells. Let us now analyze the time complexity of computing these functions. For simplicity, we use the cost function $C(x) = (x-\theta)^2$ for the multi-level cell technology as an example, but the results can be easily extended for both general cost functions in Definition 12.

When $C(x) = (x-\theta)^2$, the function $\mathcal{A}(x;1)$ is a degree-2 polynomial of x in $O(\frac{\theta}{\Delta\delta})$ intervals. By induction (for simplicity we only present the conclusion and skip the detailed analysis), for $i = 2, 3, \cdots, t$ and $j = 0, 1, \cdots, \lceil \frac{\theta}{\Delta(1-\epsilon)} \rceil$, the func-

[2]Since $\theta \leq \mathcal{L}$, in the above computation, we let $\theta = \mathcal{L}$. Functions $\mathcal{A}(x;i)$ and $\alpha(x;i;j)$ computed this way can be used for any $\theta \leq \mathcal{L}$.

tion $\alpha(x;i;j)$ is a degree-$(i+1)$ polynomial of x in $O(\frac{1-\epsilon}{\delta}(\frac{2\theta}{\Delta(1-\epsilon)})^{i-1}(\frac{\theta}{\Delta(1-\epsilon)})^{2(i-2)}i!)$ intervals; for $i = 2, 3, \cdots, t$, the function $\mathcal{A}(x;i)$ is a degree-$(i+1)$ polynomial in $O(\frac{\theta}{\Delta\delta}(\frac{2\theta}{\Delta(1-\epsilon)})^{i-1}(\frac{\theta}{\Delta(1-\epsilon)})^{2(i-1)}(i+1)!)$ intervals. So the overall time complexity of computing all the functions is $O(\frac{1-\epsilon}{\delta}(\frac{2\mathcal{L}^3}{\Delta^3(1-\epsilon)^3})^t(t+1)!)$. So when the number of rounds of charge injection t is a constant, Δ is not arbitrarily small and ϵ is not arbitrarily close 1, the complexity is upper bounded by a polynomial of the parameters. We note that the above complexity is derived based on a very pessimistic analysis. The actual complexity is usually (significantly) lower.

E. Numerical Computation

We demonstrate the numerical computation of the functions $\mathcal{A}(x;i)$ and $\alpha(x;i;j)$. We consider two cases for the cost function: for MLC, and for rank modulation (see Definition 12).

1. Multi-level Cells

For MLC, we set the cost function as

$$C(x) = (x-\theta)^2$$

and set the parameters as $\Delta = 1, \epsilon = 0.4, \delta = 0.6$.

The function $\mathcal{A}(x;i)$ is shown in Fig. 2, for $x \in [-6, 1)$ and $i = 1, 2, \cdots, 5$. We can see that $\mathcal{A}(x;i)$ is piecewise polynomial, and it monotonically decreases when i increases (because more rounds of charge injection leads to more accurate programming). We can also see that $\mathcal{A}(x;i)$ converges quickly as i increases.

As an example, we show the numerical functions of $\mathcal{A}(x;3)$ and $\alpha(x;3;3)$ in Fig. 3 and Fig. 4, respectively, for $x \in [-6, 1)$. The left column of the table shows

the domain for x, and the right column shows the polynomial ($\mathcal{A}(x;3)$ or $\alpha(x;3;3)$) in this domain.

2. Rank Modulation

For rank modulation, we set the cost function as

$$C(x) = \begin{cases} \infty & , \text{ if } x < \theta \\ x - \theta & , \text{ if } x \geq \theta \end{cases}$$

and set the parameters as $\Delta = 1, \epsilon = 0.4, \delta = 0.6$.

The function $\mathcal{A}(x;i)$ is shown in Fig. 5, for $x \in [-6, 1)$ and $i = 1, 2, \cdots, 5$. Again, we see that $\mathcal{A}(x;i)$ is piecewise polynomial, it monotonically decreases with i, and it converges quickly with i. For illustration, we also show the numerical functions of $\mathcal{A}(x;3)$ and $\alpha(x;3;3)$ in Fig. 6 and Fig. 7, respectively, for $x \in [-6, 1)$.

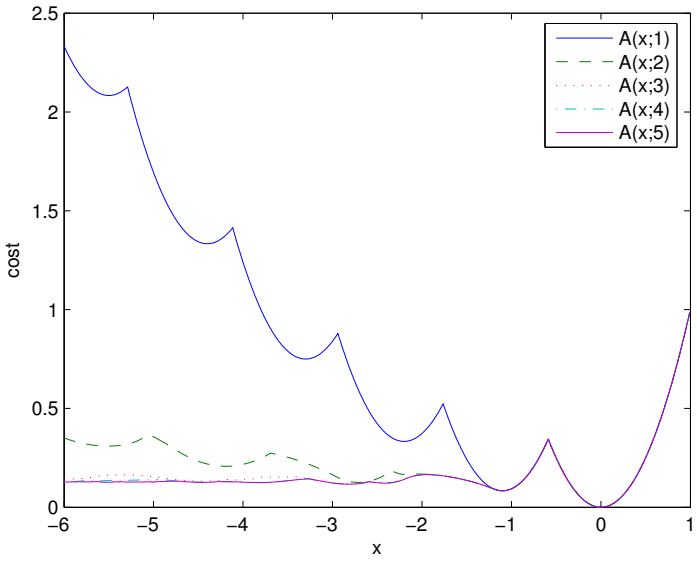

Fig. 2. The functions $\mathcal{A}(x;1)$, $\mathcal{A}(x;2)$, $\mathcal{A}(x;3)$, $\mathcal{A}(x;4)$ and $\mathcal{A}(x;5)$. Here the cost function is for MLC, and $\Delta = 1, \epsilon = 0.4, \delta = 0.6$.

$\mathcal{A}(x;3)$	
[-6,-5.56)	$23.9 + 11.2x + 1.76x^2 + 0.0917x^3$
[-5.56,-5.49)	$16.4 + 8.37x + 1.43x^2 + 0.0815x^3$
[-5.49,-5.39)	$24.9 + 12.7x + 2.16x^2 + 0.122x^3$
[-5.39,-4.76)	$14.3 + 8.76x + 1.8x^2 + 0.122x^3$
[-4.76,-4.18)	$7.66 + 4.6x + 0.924x^2 + 0.0611x^3$
[-4.18,-4.16)	$15.5 + 10.6x + 2.42x^2 + 0.183x^3$
[-4.16,-3.79)	$8.84 + 5.8x + 1.27x^2 + 0.0917x^3$
[-3.79,-3.77)	$0.948 + 1.63x + 0.722x^2 + 0.0917x^3$
[-3.77,-3.56)	$1.55 + 0.771x + 0.107x^2$
[-3.56,-3.42)	$-6.75 - 6.21x - 1.85x^2 - 0.183x^3$
[-3.42,-3.39)	$-1.22 - 0.743x - 0.1x^2 - 1.49e - 08x^3$
[-3.39,-2.59)	$5.91 + 5.57x + 1.76x^2 + 0.183x^3$
[-2.59,-2.19)	$7.1 + 7.92x + 2.97x^2 + 0.367x^3$
[-2.19,-1.82)	$1.84 + 3.1x + 1.87x^2 + 0.367x^3$
[-1.82,-1.19)	$-0.259 - 0.413x - 0.1x^2$
[-1.19,-0.588)	$1.29 + 2.2x + x^2$
[-0.588,1)	x^2

Fig. 3. The function $\mathcal{A}(x;3)$ for MLC.

$\alpha(x;3;3)$	
[-6,-5.99)	$-9.86 - 4.78x - 0.761x^2 - 0.0407x^3$
[-5.99,-5.49)	$16.4 + 8.37x + 1.43x^2 + 0.0815x^3$
[-5.49,-5.39)	$24.9 + 12.7x + 2.16x^2 + 0.122x^3$
[-5.39,-4.76)	$14.3 + 8.76x + 1.8x^2 + 0.122x^3$
[-4.76,-4.13)	$7.66 + 4.6x + 0.924x^2 + 0.0611x^3$
[-4.13,-3.99)	$5.06 + 2.29x + 0.267x^2 - 9.93e - 09x^3$
[-3.99,-2.99)	$12.8 + 8.12x + 1.73x^2 + 0.122x^3$
[-2.99,-2.39)	$9.55 + 4.85x + 0.633x^2$
[-2.39,1)	$11.6 + 6.6x + x^2$

Fig. 4. The function $\alpha(x;3,3)$ for MLC.

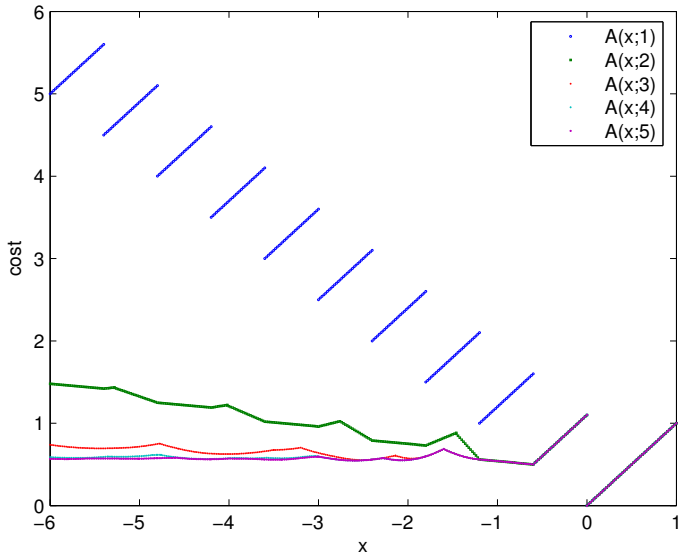

Fig. 5. The functions $\mathcal{A}(x;1)$, $\mathcal{A}(x;2)$, $\mathcal{A}(x;3)$, $\mathcal{A}(x;4)$ and $\mathcal{A}(x;5)$. Here the cost function is for rank modulation, and $\Delta = 1, \epsilon = 0.4, \delta = 0.6$.

$\mathcal{A}(x;3)$	
[-6,-5.4)	$4.76 + 1.5x + 0.138x^2$
[-5.4,-5.16)	$3.42 + 1x + 0.0917x^2$
[-5.16,-4.8)	$6.47 + 2.18x + 0.206x^2$
[-4.8,-4.77)	$4.89 + 1.52x + 0.138x^2$
[-4.77,-4.56)	$2.04 + 0.853x + 0.122x^2$
[-4.56,-4.2)	$5.22 + 2.25x + 0.275x^2$
[-4.2,-3.6)	$3.6 + 1.48x + 0.183x^2$
[-3.6,-3.5)	$2.41 + 0.817x + 0.0917x^2$
[-3.5,-3.2)	$4.08 + 1.94x + 0.275x^2$
[-3.2,-3)	$2.32 + 1.38x + 0.275x^2$
[-3,-2.66)	$1.08 + 0.56x + 0.138x^2$
[-2.66,-2.4)	$4.02 + 2.76x + 0.55x^2$
[-2.4,-2.14)	$2.43 + 1.44x + 0.275x^2$
[-2.14,-2.06)	$1.25 + 0.89x + 0.275x^2$
[-2.06,-1.8)	$4.77 + 4.3x + 1.1x^2$
[-1.8,-1.6)	$2.99 + 2.32x + 0.55x^2$
[-1.6,-1.2)	$1.23 + 1.22x + 0.55x^2$
[-1.2,-1.2)	$-0.88 - 1.2x$
[-1.2,-0.6)	$0.44 - 0.1x$
[-0.6,0)	$1.1 + x$
[0,1)	x

Fig. 6. The function $\mathcal{A}(x;3)$ for rank modulation.

$\alpha(x;3;3)$	
[-6,-5.82)	$-1.99 - 0.76x - 0.0458x^2$
[-5.82,-5.4)	$1.64 + 0.487x + 0.0611x^2$
[-5.4,-4.8)	$5.21 + 1.81x + 0.183x^2$
[-4.8,-4.56)	$2.04 + 0.853x + 0.122x^2$
[-4.56,-4.2)	$5.22 + 2.25x + 0.275x^2$
[-4.2,-3.6)	$3.6 + 1.48x + 0.183x^2$
[-3.6,-3.26)	$2.41 + 0.817x + 0.0917x^2$
[-3.26,-3)	$5.35 + 2.61x + 0.367x^2$
[-3,-2.4)	$3.7 + 1.51x + 0.183x^2$
[-2.4,-1.8)	$2.64 + 0.633x$
[-1.8,1)	$3.3 + x$

Fig. 7. The function $\alpha(x;3,3)$ for rank modulation.

CHAPTER IV

CONSTRAINED CODES FOR PHASE-CHANGE MEMORIES

Phase-change memories (PCMs) are one of the most promising candidates for the next-generation non-volatile memories. A PCM cell can have $q \geq 2$ levels, where the level 0 is the amorphous state, the levels $1, \ldots, q-2$ are the partially crystalline states, and the level $q-1$ is the crystalline state. Among them, the levels $0, 1, \ldots, q-2$ can be seen as semi-stable states, while the level $q-1$ can be seen as a stable state. The level of a PCM cell is switched by high temperatures. In the Introduction section, we have introduced two potential thermal issues for PCMs: the *thermal crosstalk problem*, and the *local thermal accumulation problem*. In this chapter, we consider coding techniques for these potential challenges. For the thermal crosstalk problem, we use a scheme that removes the crosstalk interference, and then study coding techniques that reduce the programming cost (measured by the number of RESET operations). For the local thermal accumulation problem, we study coding techniques that impose time and space constraints on writing, to help the heat generated by programming be more balanced spatially.

A. Symbol-constrained Codes

In this section, we study coding techniques for the thermal crosstalk problem. Let $c_1, c_2, \ldots, c_n$ be n cells. For $i \in \{1, 2, \ldots, n\} \triangleq [n]$, let $\ell_i \in \{0, 1, \ldots, q-1\}$ denote the level of c_i. In this paper, we consider the cells as a one-dimensional array. (The concepts can be extended to higher dimensions, too.) Two cells c_i and c_j are neighbors if and only if $|i-j| = 1$. Let $\gamma \in \{1, 2, \ldots, q-1\}$ be a parameter.

We consider the following simple model for thermal crosstalk: *When a cell c_i is RESET to level 0, the thermal crosstalk from c_i (at that moment) will increase its*

neighboring cell' level ℓ_j (for $j = i \pm 1$) by at most γ, unless the neighboring cell c_j is also being RESET at that moment. (However, a cell level cannot exceed $q-1$, the stable fully-crystalline state. And a SET operation does not affect the neighboring cells due to its considerably lower temperature.)

Let $\mathcal{C} \subseteq \{0, 1, \ldots, q-1\}^n$ be a code, whose rate $R(\mathcal{C})$ is defined as $\frac{\log_2|\mathcal{C}|}{n}$. (Clearly, $R(\mathcal{C}) \leq \log_2 q$.) A *rewrite* is to change the cell levels from the current codeword $X = (x_1, \ldots, x_n) \in \mathcal{C}$ to a new codeword $Y = (y_1, \ldots, y_n) \in \mathcal{C}$. For $i \in [n]$, if $x_i > y_i$, then the rewrite needs to RESET c_i (and then SET c_i if $y_i > 0$). For $j = i \pm 1$, if c_i is RESET and $y_j < \min\{x_j + \gamma, q-1\}$, then the rewrite needs to RESET c_j as well, because otherwise the thermal crosstalk from c_i may make ℓ_j greater than y_j. Therefore, a RESET operation applied to a cell can trigger the RESET of its neighboring cell, and this effect can propagate to many cells. Let us define a *RESET segment* in codeword Y as a maximum run of symbols $y_i, y_{i+1}, \ldots, y_j$ (where $1 \leq i \leq j \leq n$) such that: (1) $\forall\, i' \in \{i, \ldots, j\}$, $y_{i'} < \min\{x_{i'} + \gamma, q-1\}$; (2) $\exists\, i'' \in \{i, \ldots, j\}$ such that $x_{i''} > y_{i''}$. By our above analysis, the rewrite must RESET all the cells in a RESET segment (before setting them).

To rewrite cells using parallel programming, it is natural to use the following two-step procedure: *First, RESET all the cells in RESET segments of the new codeword; then, SET all the cells whose levels are still lower than their values in the new codeword.* (The second step has no crosstalk effect.)

Example 18.. *Let $n = 11$, $q = 4$ and $\gamma = 3$. Assume the cells need to change from an old codeword $(1, 3, 2, 2, 2, 2, 2, 2, 1, 1, 1)$ to a new codeword $(0, 3, 2, 2, 1, 2, 2, 2, 3, 1, 2)$. First, we RESET the cells $\{c_1, c_3, c_4, c_5, c_6, c_7, c_8\}$. (After this step, the cell levels will be $(0, 3, 0, 0, 0, 0, 0, 0, \ell_9, 1, 1)$, where $\ell_9 \in \{1, 2, 3\}$. (Since cell c_8 is RESET, the thermal crosstalk from c_8 may make ℓ_9 be greater than its original value*

1.) Then, we SET the cells $\{c_3, c_4, c_5, c_6, c_7, c_8, c_9, c_{11}\}$*, to increase the cell levels to* $(0, 3, 2, 2, 1, 2, 2, 2, 3, 1, 2)$.

We define the *cost* of a rewrite operation as the number of cells that are RESET during rewriting. (In Example 18, the cost is 7.) The number of RESETs is a very important cost measurement because PCM cells have a limited longevity: PCM cells can endure about $10^6 \sim 10^8$ RESETs (or SET-RESET cycles) before becoming non-functional [3, 16]. Note that for the rewrite, for every cell that needs to decrease its level, the whole RESET segment containing it is forced (triggered) to be RESET, too. This motivates us to study constrained codes where RESET segments have *limited lengths*. Given this constraint, we seek capacity-achieving codes.

In this paper, we focus on the case $\gamma = q-1$ (the worst-case scenario for thermal crosstalk). We define an *unstable segment* in a codeword $X = (x_1, \ldots, x_n)$ as a maximum run of symbols $x_i, x_{i+1}, \ldots, x_j$ (where $1 \leq i \leq j \leq n$) such that for $i' \in \{i, \ldots, j\}$, $x_{i'} < q-1$. The *length* of this unstable segment is $j - i + 1$. When the memory writes X, an unstable segment in it will become a RESET segment if any of the cells in that unstable segment needs to decrease its level (compared to the old codeword). For a code $\mathcal{C}$, if in all its codewords the unstable segments' lengths are at most k, then during rewriting, the length of every RESET segment is at most k.

Definition 19.. SYMBOL-CONSTRAINED CODES

Let k *be a positive integer. A code* $\mathcal{C} \subseteq \{0, 1, \ldots, q-1\}^n$ *is* k*-limited if in every codeword of* $\mathcal{C}$*, every unstable-segment's length is at most* k*. (It is also called a symbol-constrained code.)*

The k-limited codes are a constrained system $\mathcal{S}$ over alphabet $\Sigma \triangleq \{0, 1, \ldots, q-1\}$. An example for $q = 4, k = 3$ is shown in Fig. 8. We see that it generalizes the $(d = 0, k)$-run-length-limited (RLL) codes [18] from the binary alphabet to the q-ary

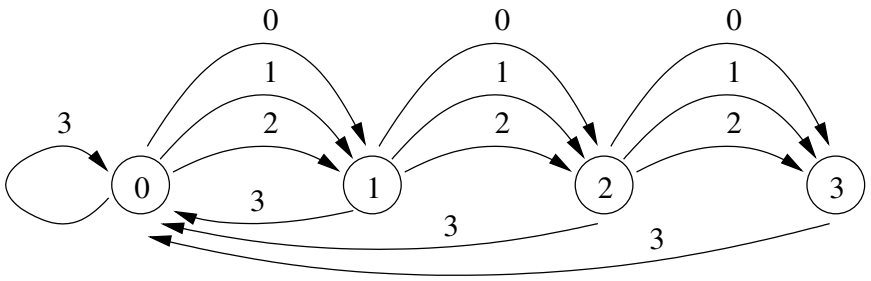

Fig. 8. Shannon cover of the 3-limited codes (constrained system), for $q = 4$.

alphabet. Its Shannon capacity is $cap(\mathcal{S}) = \lim_{n\to\infty} \sup \frac{1}{n} \log N(n; \mathcal{S})$, where $N(n; \mathcal{S})$ is the number of words of length n in $\mathcal{S}$. We have constructed a k-limited code for $q = 4$ and $k = 1$, which has a rate $6 : 5$ finite-state encoder. Its rate is 1.2 bits/cell, close to the Shannon capacity (which can be shown to be 1.203). When the code is used for storing data, assuming that the input information bits have a uniform i.i.d. distribution, for every rewrite, the ratio of the average number of RESET operations to the number of information bits is 0.228. This compares favorably with the no-coding method (i.e., storing 2 bits per cell), which has a higher ratio of 0.345. So symbol-constrained codes can reduce RESETs.

We note that rewriting codes for reducing the RESET operations for PCMs have been studied in [16], where interesting WOM-like codes have been used. However, the study in [16] did not consider any thermal interference problem. We also stress that the codes in [16] and the codes we study are two drastically different approaches. While the codes in [16] always RESET all cells at the same time (to get a fresh start for rewriting), we use constrained codes that are based on local constraints, and cells are most likely RESET in different rewrites. And with our constrained-coding approach, slide-block decoders can be built to locally decode information bits efficiently.

The following theorem presents the Shannon capacity of the symbol-constrained codes, for arbitrary q and k.

Theorem 20.. *Let $q \geq 2$ and $k \geq 1$ be integers. Let*

$$f(\lambda) = \lambda^{k+2} - q\lambda^{k+1} + (q-1)^{k+1}.$$

The equation $f(\lambda) = 0$ has at most three real-valued solutions, one of which is $q-1$. Among those real-valued solutions, if $q = k+2$, let λ^* be the solution with the greatest absolute value; otherwise, among the (at most two) real-valued solutions unequal to $q-1$, let λ^* be the solution with the greater absolute value. Then the Shannon capacity of k-limited codes is

$$\log_2 |\lambda^*|$$

bits per cell.

Proof. Let $A_{q,k}$ denote the adjacency matrix of the Shannon cover of the k-limited codes in cells of q levels. (See Fig. 8 for an example.) We have

$$\begin{aligned} A_{q,k} &= \begin{pmatrix} 1 & q-1 & 0 & \cdots & 0 \\ 1 & 0 & q-1 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & 0 & 0 & \cdots & q-1 \\ 1 & 0 & 0 & \cdots & 0 \end{pmatrix} \\ &= \begin{pmatrix} 1_{k\times 1} & (q-1)E_k \\ 1 & 0_{1\times k} \end{pmatrix} \end{aligned}$$

Here $1_{k\times 1}$ denotes the all-one column-vector of length k, E_k denotes the $k \times k$ identity matrix, and $0_{1\times k}$ denotes the all-zero row vector of length k. $A_{q,k}$ is a $(k+1) \times (k+1)$ matrix.

Define $B_{q,k}$ (for $k \geq 2$) as a $k \times k$ matrix as follows:

$$B_{q,k} = A_{q,k-1} - \lambda E_k + \begin{pmatrix} \lambda & 0 & \cdots & 0 \\ 0 & 0 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 0 \end{pmatrix}_{k \times k}$$

Let us show by induction that $|B_{q,k}| = (-1)^{k+1} \sum_{i=0}^{k-1} (q-1)^{k-1-i} \lambda^i$. When $k = 2$, we have $|B_{q,2}| = \begin{vmatrix} 1 & q-1 \\ 1 & -\lambda \end{vmatrix} = -\lambda - (q-1)$. This serves as the base case. Now consider $k \geq 3$. We have $|B_{q,k}| = (-\lambda)^{k-1} - (q-1)\,|B_{q,k-1}|$. By the induction assumption, we get $|B_{q,k}| = (-1)^{k+1} \sum_{i=0}^{k-1} (q-1)^{k-1-i} \lambda^i$.

Let $\mathcal{S}$ denote the constrained system (the k-limited codes), and let λ^* denote the eigenvalue of the greatest absolute value of the matrix $A_{q,k}$. The capacity of $\mathcal{S}$ equals $\log_2 |\lambda^*|$ bits per cell. To find λ^*, we need to compute $|A_{q,k} - \lambda E_{k+1}|$. ($E_{k+1}$ is the $(k+1) \times (k+1)$ identity matrix.) First, consider $k = 1$. We get

$$|A_{q,1} - \lambda E_2| = \begin{vmatrix} 1-\lambda & q-1 \\ 1 & -\lambda \end{vmatrix} = \lambda^2 - \lambda - (q-1)$$

By solving $|A_{q,1} - \lambda E_2| = 0$, we get $\lambda^* = \frac{1+\sqrt{1+4(q-1)}}{2}$. Since when $k = 1$, we have $f(\lambda) = \lambda^3 - q\lambda^2 + (q-1)^2 = (\lambda - (q-1))(\lambda^2 - \lambda - (q-1)) = (\lambda - (q-1))\,|A_{q,1} - \lambda E_2| = (\lambda - (q-1))(\lambda - \frac{1+\sqrt{1+4(q-1)}}{2})(\lambda - \frac{1-\sqrt{1+4(q-1)}}{2})$, it is not difficult to verify that the theorem holds for $k = 1$.

Let us show that the theorem holds for $k \geq 2$, too. When $k \geq 2$, we get

$$\begin{aligned} & |A_{q,k} - \lambda E_{k+1}| \\ = \; & (1-\lambda)(-\lambda)^k - (q-1)\,|B_{q,k}| \\ = \; & \frac{(-1)^{k+1}\left(\lambda^{k+2} - q\lambda^{k+1} + (q-1)^{k+1}\right)}{\lambda - (q-1)} \\ = \; & \frac{(-1)^{k+1}}{\lambda-(q-1)} \cdot f(\lambda) \end{aligned}$$

So the equation $f(\lambda) = 0$ has at most one more real solution compared to the equation $|A_{q,k} - \lambda E_{k+1}| = 0$ (which would be the solution $\lambda = q-1$).

To see that $f(\lambda) = 0$ has at most three real-valued solutions, consider $f'(\lambda) = (k+2)\lambda^{k+1} - q(k+1)\lambda^k$. Since $f'(\lambda) = 0$ has only two solutions ($\lambda = 0$ and $\lambda = \frac{q(k+1)}{k+2}$), $f(\lambda)$ is (strictly) monotonic in the three ranges for λ: $(-\infty, 0], [0, \frac{q(k+1)}{k+2}], [\frac{q(k+1)}{k+2}, \infty)$. Therefore $f(\lambda) = 0$ has at most three real-valued solutions.

We now show that $\lambda = q-1$ is a solution to $|A_{q,k} - \lambda E_{k+1}| = 0$ if and only if $q = k+2$. By replacing λ with $q-1$ in $|A_{q,k} - \lambda E_{k+1}|$, we get

$$|A_{q,k} - \lambda E_{k+1}| = (-1)^{k+1}(q-1)^k(q-k-2),$$

which equals 0 if and only if $q = k+2$.

So we can see that the solution λ^* defined in the theorem is the solution of the greatest absolute value to the equation $|A_{q,k} - \lambda E_{k+1}| = 0$. Therefore $|\lambda^*|$ is the largest of the absolute values of the eigenvalues of $A_{q,k}$. So the theorem holds for $k \geq 2$, too. □

Based on Theorem 20, the Shannon capacity of symbol-constrained codes for different q and k are shown in Table I.

Table I. Shannon capacity (bits per cell) of k-limited codes

$q\backslash k$	1	2	3	4	5	6
2	0.694	0.879	0.947	0.975	0.988	0.994
3	1.000	1.303	1.432	1.496	1.531	1.552
4	1.203	1.585	1.756	1.846	1.899	1.931
5	1.357	1.797	2.000	2.110	2.176	2.218
6	1.481	1.968	2.196	2.322	2.399	2.449
7	1.585	2.111	2.360	2.499	2.585	2.642
8	1.675	2.234	2.501	2.651	2.745	2.807
9	1.754	2.342	2.624	2.785	2.885	2.953
10	1.824	2.438	2.734	2.904	3.010	3.082
11	1.888	2.525	2.834	3.011	3.123	3.199
12	1.946	2.604	2.924	3.108	3.225	3.305
13	2.000	2.677	3.008	3.198	3.319	3.402
14	2.050	2.745	3.084	3.281	3.406	3.492
15	2.096	2.807	3.156	3.358	3.487	3.576
16	2.139	2.866	3.223	3.430	3.563	3.654

B. Space-time Constrained Codes

In this section, we study coding techniques for a different interference problem: the local thermal accumulation problem. It is known that when cells are repeatedly programmed, adjacent cells can be crystallized/disturbed [19]. We seek codes for rewriting data that can balance heat better. This motivates us to study the space-time constrained codes defined below.

Let $c_1, \ldots, c_n$ be n PCM cells, whose levels are denoted by $\ell_1, \ldots, \ell_n \in \{0, \ldots, q-1\}$. Let $V = \{0, 1, \ldots, v-1\}$ be an alphabet of size v. The data stored in the n cells takes its value from the alphabet V. A code $\mathcal{C}$ is a mapping from the cell levels $L \triangleq (\ell_1, \ldots, \ell_n) \in \{0, \ldots, q-1\}^n$ to the data values V. We allow it to be a many-to-one mapping (instead of a one-to-one mapping). The code $\mathcal{C}$ has two associated functions: a *decoding function* F_d and an *update function* F_u. The decoding function $F_d : \{0, \ldots, q-1\}^n \to V$ tells us that the cell levels L represent the data $F_d(L) \in V$. The update function $F_u : \{0, \ldots, q-1\}^n \times V \to \{0, \ldots, q-1\}^n$ tells us that if the old cell levels are L and we want to write the new data $s \in V$ into the cells, we will change the cell levels to $F_u(L, s)$. (Clearly, we should have $F_d(F_u(L, s)) = s$.) A rewrite can change the data to any value in V. Here we do not consider the thermal crosstalk problem. So when a rewrite changes an old codeword $X = (x_1, \ldots, x_n) \in \{0, \ldots, q-1\}^n$ to a new codeword $Y = (y_1, \ldots, y_n)$, for $i \in [n]$, a cell c_i needs to be programmed only if $x_i \neq y_i$. We define the *rewrite cost* as the number of cells that are programmed, $|\{i \in [n] \mid x_i \neq y_i\}|$, which is the Hamming distance between X and Y. To balance programming-generated heat, we study the following code.[1]

[1]The model can be generalized by differentiating the cost of RESET and SET operations. In PCMs, the RESET operation uses a higher temperature that the SET operation, but has a shorter duration of time.

Definition 21.. Space-time constrained codes

Let α, β, p be positive integers. A code is (α, β, p)-constrained if for any α consecutive rewrites and for any segment of β cells – namely, $c_i, c_{i+1}, \ldots, c_{i+\beta-1}$ for some $i \in [n]$ – the total rewrite cost of those β cells (over those α rewrites) is at most p. (It is also called a space-time constrained code.)

We note that the space-time constrained codes are interesting because although the system can keep moving data that are frequently rewritten to balance heat, such an approach may cause substantial overhead for file-system/compiler design and their optimization. And for content-addressable systems, where the address of data is determined by the content of the data (e.g., by using a hash function) for fast data retrieval, relocating data can also be very challenging. In this work, as the starting point of understanding space-time constrained codes, we study the time and space constraints separately.

1. Time-constrained Codes

We first study time-constrained codes with $\alpha \geq 1, \beta = 1, p = 1$. This is the simple case where every cell can be programmed at most once in every α consecutive rewrites. Note that the rate of the code $\mathcal{C}$ is defined as $\frac{\log_2 v}{n}$ bits per cell. It is easy to see that a simple idea based on time division can give us a code of rate $\frac{\log_2 q}{\alpha}$ bits per cell, as follows: Let $n = \alpha\lceil \log_q v \rceil$, and divide the n cells evenly into α groups (call them the 0th, 1st, 2nd, …, $(\alpha-1)$-th cell groups); for $i = 1, 2, 3 \cdots$, for the i-th rewrite we write the data into the $(i \mod \alpha)$-th cell group. When $n \to \infty$ (which also means $v \to \infty$), the code rate approaches $\frac{\log_2 q}{\alpha}$ bits per cell. So the question is if there exist codes of higher rates.

Note that the challenge for designing time-constrained codes is that we cannot

Table II. WOM code D with t=2

L'	000	100	010	001	110	101	011	111
$F_{d(\mathcal{D})}(L')$	0	1	2	3	3	2	1	0

afford to remember for every cell how long ago the cell was programmed for the last time (up to α past rewrites), because that alone will cost $\log_2 \alpha$ bits of storage space for every cell. (Consider the case $q = 2$.) So the programming of cells needs to be synchronized in some way so that this information cost can be reduced. We now present a general time-constrained code construction for $q = 2$ that uses the write-once memory (WOM) codes [22] as sub-codes.

Let $\mathcal{D}$ be a WOM code that stores data of alphabet size w in m cells of $q = 2$ levels. Denote the alphabet of the stored data by $W = \{0, 1, \ldots, w-1\}$. The code $\mathcal{D}$ also has a *decoding function* $F_{d(\mathcal{D})} : \{0,1\}^m \to W$ and an *update function* $F_{u(\mathcal{D})} : \{0,1\}^m \times W \to \{0,1\}^m$. WOM codes have a unique property: *with every rewrite, the cell levels can only increase, not decrease* [22]. Let t denote the number of rewrites the code $\mathcal{D}$ can guarantee to support. (Let the initial cell levels all be zero.) Clearly, due to the unique property of WOM codes, t is a finite number.

Example 22.. *Let $w = 4$, $m = 3$, $q = 2$. Let $L' \triangleq (\ell'_1, \ell'_2, \ell'_3) \in \{0,1\}^3$ denote the three cell levels. The following WOM code $\mathcal{D}$ was presented by Rivest and Shamir [22] with $t = 2$ in Table II.*

If the $t = 2$ rewrites first write the data as 2, the rewrite it as 1, the code will first let L' be $(0, 1, 0)$, the change it to $(0, 1, 1)$.

Let $\mathcal{E}$ be an "elevator code" that mimics $\mathcal{D}$ but allows the cell levels to increase and decrease in a synchronized way, described as follows. $\mathcal{E}$ also stores data of alphabet size w in m cells of $q = 2$ levels. Plainly speaking, for the first α rewrites, $\mathcal{E}$ rewrites data in the same way as $\mathcal{D}$; then it pushes all the m cell levels to $q - 1 = 1$;

for the next α rewrites, $\mathcal{E}$ rewrites data by decreasing cell levels, in exactly the opposite way of $\mathcal{D}$; then it pushes all the m cell levels to 0; then the third batch of α rewrites are implemented in the same way as $\mathcal{D}$ again; and so on. We now formally define the *decoding function* $F_{d(\mathcal{E})}$ and the *update function* $F_{u(\mathcal{E})}$ of $\mathcal{E}$. Let us call a sequence of rewrites the 0th, 1st, 2nd, 3rd $\cdots$ rewrites. For $i = 0, 1, 2 \ldots$, let L'_i denote the cell levels after the i-th rewrite, and let $e_i \in W$ denote the data that the i-th rewrite writes into the cells. (Clearly, we should have $F_{d(\mathcal{E})}(L'_i) = e_i$.) Then if $0 \leq (i \mod 2\alpha) \leq \alpha - 1$,

$$F_{d(\mathcal{E})}(L'_i) = F_{d(\mathcal{D})}(L'_i);$$

otherwise,

$$F_{d(\mathcal{E})}(L'_i) = F_{d(\mathcal{D})}((q-1, \cdots, q-1) - L'_i).$$

If $i \equiv 0 \mod 2\alpha$,

$$F_{u(\mathcal{E})}\left(L'_{i-1}, e_i\right) = F_{u(\mathcal{D})}((0, \cdots, 0), e_i).$$

If $1 \leq (i \mod 2\alpha) \leq \alpha - 1$,

$$F_{u(\mathcal{E})}\left(L'_{i-1}, e_i\right) = F_{u(\mathcal{D})}\left(L'_{i-1}, e_i\right).$$

If $i \equiv \alpha \mod 2\alpha$,

$$F_{u(\mathcal{E})}\left(L'_{i-1}, e_i\right) = (q-1, \ldots, q-1) - F_{u(\mathcal{D})}((0, \cdots, 0), e_i).$$

If $\alpha + 1 \leq (i \mod 2\alpha) \leq 2\alpha - 1$,

$$F_{u(\mathcal{E})}\left(L'_{i-1}, e_i\right) = (q-1, \ldots, q-1) - F_{u(\mathcal{D})}\left((q-1, \cdots, q-1) - L'_{i-1}, e_i\right).$$

Example 23.. *Let $\mathcal{D}$ be the WOM code in Example 22, and let $\mathcal{E}$ be the "elevator code" defined as above. Then when the rewrites change the data as $1 \to 2 \to 3 \to 2 \to 3 \to 1 \to \cdots$, the code $\mathcal{E}$ changes the cell levels as $(0,0,0) \to (1,0,0) \to (1,0,1) \to$*

$$(1,1,1) \to (1,1,0) \to (0,1,0) \to \ (0,0,0) \to (0,0,1) \to (0,1,1) \to \ (1,1,1) \to \cdots$$

We now construct the code $\mathcal{C}$ that stores data of alphabet size v in n cells of $q = 2$ levels. Let $v = w^{\frac{t}{\gcd(t,\alpha)}}$, where $\gcd(t,\alpha)$ is the greatest common divisor of t and α. Let $n = \frac{m(t+\alpha)}{\gcd(t,\alpha)}$. We see the stored data as a vector $X = \left(x_1, x_2, \ldots, x_{\frac{t}{\gcd(t,\alpha)}}\right) \in \{0,1,\ldots,w-1\}^{\frac{t}{\gcd(t,\alpha)}}$. For $i = 0,1,2\cdots$, let X_i denote the data (vector) that the i-th rewrite writes into the n cells. We divide the n cells evenly into $\frac{t+\alpha}{\gcd(t,\alpha)}$ groups, and call them the 0th, 1st, $\cdots$, $(\gcd(t,\alpha)-1)$-th groups. (Every cell group has m cells.) We implement a sequence rewrites as follows. For $i = 0,1,2\ldots$, let $g(i) = \lfloor \frac{i}{\gcd(t,\alpha)} \rfloor$. Then for the i-th rewrite, the $\frac{t}{\gcd(t,\alpha)}$ elements of X_i are, respectively, written into the $\left(g(i) \mod \frac{t+\alpha}{\gcd(t,\alpha)}\right)$-th, $\left(g(i)+1 \mod \frac{t+\alpha}{\gcd(t,\alpha)}\right)$-th, $\cdots$, $\left(g(i)+\frac{t}{\gcd(t,\alpha)}-1 \mod \frac{t+\alpha}{\gcd(t,\alpha)}\right)$-th cell groups. (After the rewrite, the data can be decoded from those cell groups as well.) Every cell group uses the "elevator code" $\mathcal{E}$ to rewrite data. (For a cell group, after it is used for t consecutive rewrites, all its cell levels will be pushed to zero or $q-1$ when the next rewrite comes. The it will rest for $\alpha - 1$ rewrites.)

It is not hard to see that every cell group will be programmed in $t+1$ consecutive rewrites, then not programmed for another $\alpha-1$ consecutive rewrites, and then repeat this process. In such a period of $t+\alpha$ rewrites, the cell levels are either all increasing or all decreasing; since $q = 2$, every cell can be programmed only once. So every cell is programmed at most once for every α consecutive rewrites. So $\mathcal{C}$ is a time-constrained code.

The only detail left to specify is how to know the value of $i \mod 2(t+\alpha)$ when the i-th rewrite happens, which is needed in the above coding process. (It is used to compute $g(i)$ and to implement the "elevator code.") This value can be obtained by using a simple "counter" of $2(t+\alpha)$ cells of $q = 2$ levels. Let $\ell'_1, \ell'_2, \ldots, \ell'_{2(t+\alpha)}$

Table III. Rates of the time-constrained codes

α		4	5	6	7	8
$1/\alpha$		0.250	0.200	0.167	0.143	0.125
rate of $\mathcal{C}$	$t=2$	0.258	0.221	0.193	0.172	0.155
	$t=3$	0.277	0.242	0.215	0.194	0.176
	$t=4$	0.280	0.249	0.224	0.204	0.187
	$t=5$	0.277	0.250	0.227	0.208	0.192

denote their levels. We cyclically program the cells; for every rewrite, we change the level of one cell. We see $\sum_{j=1}^{2(t+\alpha)-1} \left|\ell'_j - \ell'_{2(t+\alpha)}\right|$ equals $i \mod 2(t+\alpha)$. So we can get the wanted value, and every cell in the counter is programmed exact once for every $2(t+\alpha) > \alpha$ rewrites.

Let $w \to \infty$, fix t as a constant, and we choose the smallest m such the WOM code $\mathcal{D}$ exists. By the known results on WOM codes [22], when $t = 2$, $m \approx 1.294 \log_2 w$; when $t = 3$, $m \approx 1.549 \log_2 w$; $\cdots$; for sufficiently large t, $m \approx \frac{t}{\log_2 t} \cdot \log_2 w$. The rate of the time-constrained code $\mathcal{C}$ is $\lim_{w\to\infty} \frac{\log_2 v}{n+2(t+\alpha)} = \lim_{w\to\infty} \frac{\frac{t}{\gcd(t,\alpha)} \log_2 w}{\frac{m(t+\alpha)}{\gcd(t,\alpha)}} = \frac{t}{t+\alpha} \cdot \frac{\log_2 w}{m}$. By the known values of $\frac{\log_2 w}{m}$ [22], we show the rate of $\mathcal{C}$ in the following table, and compare it with $\frac{1}{\alpha}$ (the rate of the code using time sharing). We see that the code $\mathcal{C}$ can achieve a higher rate in Table III.

The following theorem presents an upper bound to the rate of time-constrained codes.

Theorem 24.. *Define* v_{max} *as*

$$v_{max} \triangleq \max_{\Delta=1,2,\ldots,\lfloor \frac{n}{\alpha} \rfloor} \sum_{i=0}^{\Delta} \binom{n-(\alpha-1)\Delta}{i} (q-1)^i.$$

Then the rate of $(\alpha, 1, 1)$*-constrained codes that use* n *cells of* q *levels is upper bounded by* $(\log_2 v_{max})/n$ *bits per cell.*

Proof. Let $\mathcal{C}$ be a time-constrained code that stores the data of alphabet size v in the n cells of q levels. Let $X \triangleq (x_1, x_2, x_3, \cdots)$ represent the data stored by a sequence of rewrites; that is, for $i = 1, 2, 3 \cdots$, the i-th rewrite writes the data $x_i \in \{0, 1, \ldots, v-1\}$ into the n cells. For $i = 1, 2, 3 \cdots$, let δ_i denote the number of cells programmed by the i-th rewrite. We choose X in the following greedy way: Choose $x_1, x_2, x_3 \cdots$ one by one, and each time – say we are choosing for the i-th rewrite – we choose x_i from the alphabet $\{0, 1, \ldots, v-1\}$ greedily such that δ_i is locally maximized.

We construct a sequence of positive integers $\Delta_1, \Delta_2, \Delta_3, \cdots$ in the following way. First, for convenience, let $\Delta_0 = \Delta_{-1} = \Delta_{-2} = \cdots = \Delta_{-\alpha+2} = 0$. Then, for $i = 1, 2, 3, \cdots$, let Δ_i be the smallest positive integer such that

$$v \leq \sum_{k=0}^{\Delta_i} \binom{n - \sum_{j=i-\alpha+1}^{i-1} \Delta_j}{k} (q-1)^k .$$

For convenience, let $\delta_0 = \delta_{-1} = \cdots = \delta_{-\alpha+2} = 0$. We now show by induction that $\delta_i \geq \Delta_i$ for $i \geq -\alpha + 2$. As the base case, we have $\delta_i = \Delta_i = 0$ for $i = -\alpha + 2, \cdots, 0$. Now consider $i \geq 1$. For the i-th rewrite, the number of cells that can be programmed are those cells that were not programmed in the previous $\alpha - 1$ rewrites. For the i-th rewrite, there are $n - \sum_{j=i-\alpha+1}^{i-1} \delta_j$ cells that can be programmed. Let y be the smallest integer such that

$$v \leq \sum_{k=0}^{y} \binom{n - \sum_{j=i-\alpha+1}^{i-1} \delta_j}{k} (q-1)^k .$$

There are $\sum_{k=0}^{y} \binom{n - \sum_{j=i-\alpha+1}^{i-1} \delta_j}{k} (q-1)^k$ ways to program up to y cells for the i-th rewrite (even if we do not consider how the code $\mathcal{C}$ maps the cell levels to the data); so for the i-th rewrite, there is a choice for the new data value x_i that will require the rewrite to program at least y cells. By the way the rewriting sequence X is chosen, clearly we have $y \leq \delta_i$. By the induction assumption, we have $\delta_j \geq \Delta_j$ for

$j = i-1, i-2, \cdots, i-\alpha+1$. So $n - \sum_{j=i-\alpha+1}^{i-1} \delta_j \leq n - \sum_{j=i-\alpha+1}^{i-1} \Delta_j$. By the way Δ_i and y are defined, it is easy to see that $\Delta_i \leq y$. So $\Delta_i \leq \delta_i$, which completes the induction.

It is not hard to see that since $\Delta_i \leq \delta_i$ for all i, the infinite sequence $\Delta_1, \Delta_2, \Delta_3, \cdots$ indeed exists.

We now show by induction that $\Delta_i \leq \Delta_j$ for all $-\alpha+2 \leq i < j$; that is, the sequence $\Delta_{-\alpha+2}, \cdots, \Delta_0, \Delta_1, \Delta_2, \cdots$ monotonically increases. Trivially, $\Delta_{-\alpha+2} = \cdots = \Delta_0 = 0 < \Delta_1$. Now consider $i \geq 2$, and let us compare Δ_{i-1} with Δ_i. Since $\Delta_{i-\alpha} \leq \Delta_{i-1}$, we have $n - \sum_{j=(i-1)-\alpha+1}^{(i-1)-1} \Delta_j \geq n - \sum_{j=i-\alpha+1}^{i-1} \Delta_j$; then it is simple to see that $\Delta_{i-1} \leq \Delta_i$.

Since $\Delta_i \leq n$ for all i, the monotonic sequence $\Delta_1, \Delta_2, \Delta_3, \cdots$ must converge to a certain value. That is, there exist integers j and Δ such that for all $i \geq j$, we have $\Delta_i = \Delta$. By the way Δ_i is defined, we get

$$v \leq \sum_{k=0}^{\Delta} \binom{n-(\alpha-1)\Delta}{k} (q-1)^k .$$

That leads to the final conclusion. □

2. Space-constrained Codes

We now study space-constrained codes with $\alpha = 1$, $\beta \geq 1$ and $p = 1$. This is the simple case where for every segment of β cells – namely, $c_i, c_{i+1}, c_{i+\beta-1}$ for some $i \in [n]$ – a rewrite will program at most one cell in the segment. Note that the code uses n cells of q levels to store data of alphabet size v.

We derive an upper bound for the rate of space-constrained codes. Let $\vec{x} = (x_1, x_2, \ldots, x_n) \in \{0, 1, \ldots, q-1\}^n$ be a vector that is not equal to $(0, 0, \ldots, 0)$. We call $\vec{x}$ a *β-constrained vector* if for any two non-zero entries x_i and x_j in $\vec{x}$, we have $|i-j| \geq \beta$. Let $M_{n,\beta}$ be the set of all β-constrained vectors. We see that with a

space-constrained code, if the current cell levels are $L' = (\ell'_1, \ell'_2, \ldots, \ell'_n)$, a rewrite can change it only to the cell-level states in the set $\{L' + \vec{x} \mid \vec{x} \in M_{n,\beta}\}$. (For all the entries in the vector $L' + \vec{x}$, take modulo q.) Since the stored data have v distinct values, and a rewrite can change the data from any value to any other value, we have

$$v \leq |M_{n,\beta}| + 1.$$

So for space-constrained codes that use n cells of q levels, the code rate is upper bounded by $\frac{\log_2(|M_{n,\beta}|+1)}{n}$ bits per cell.

We now compute the value of $|M_{n,\beta}|$. When $n \leq \beta$, $|M_{n,\beta}| = n(q-1)$ because only one entry in a vector $\vec{x} \in M_{n,\beta}$ can be non-zero. Now consider the case $n \geq \beta+1$. Let $\vec{x} = (x_1, \ldots, x_n)$ be a generic vector in $M_{n,\beta}$. If $x_{n-1} = x_{n-2} = \cdots = x_{n-\beta+1} = 0$, then there are $|M_{n-\beta,\beta}| \cdot q + (q-1)$ ways to choose the values of $x_1, \ldots, x_{n-\beta}$ and x_n. If one of the elements in $\{x_{n-1}, x_{n-2}, \ldots, x_{n-\beta+1}\}$ is not zero, then x_n must be zero; and it is not hard to see that in this case, the number of choices for $\vec{x}$ is $|M_{n-1,\beta}| - |M_{n-\beta,\beta}|$. So we have the recursion

$$\begin{aligned} & |M_{n,\beta}| \\ = \; & |M_{n-\beta,\beta}| \cdot q + (q-1) + (|M_{n-1,\beta}| - |M_{n-\beta,\beta}|) \\ = \; & |M_{n-1,\beta}| + (q-1)\,|M_{n-\beta,\beta}| + q - 1 \end{aligned}$$

Along with the β initial values $|M_{n,\beta}| = n(q-1)$ for $n = 1, 2, \ldots, \beta$, we can use the above recursion to solve for $|M_{n,\beta}|$.

Note that when $q = 2$, the vectors in $M_{n,\beta}$ correspond to the codewords of length n in the $(d = \beta - 1, k = \infty)$-RLL constrained system [18], except that $M_{n,\beta}$ does not contain the all-zero codeword. Therefore, when $q = 2$ and $n \to \infty$, the rate of the space-constrained code is upper bounded by the capacity of the $(\beta - 1, \infty)$-RLL constrained system.

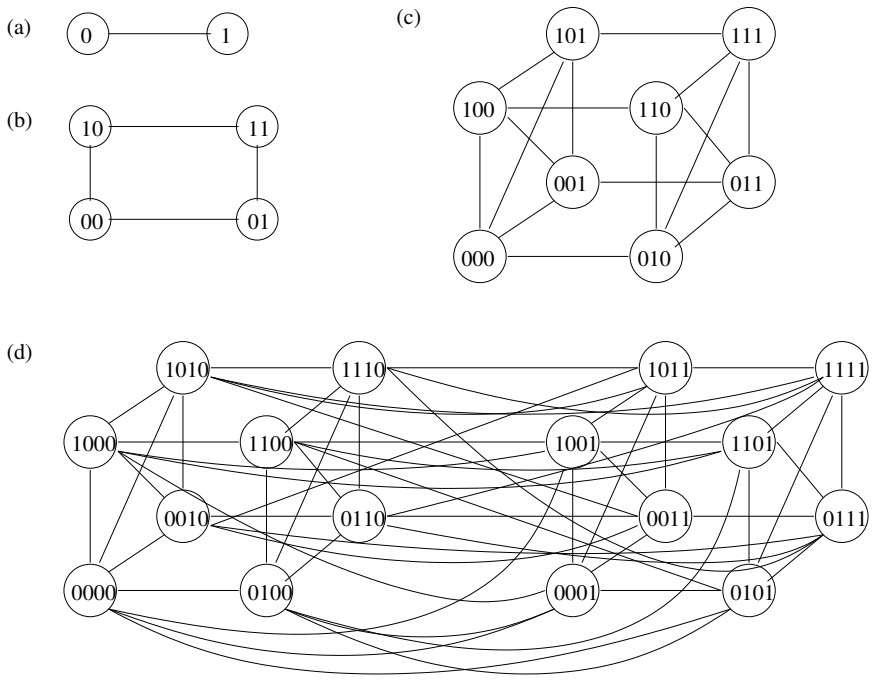

Fig. 9. $\mathcal{S}_{n,\beta}$ with $q = 2$, $\beta = 2$. (a) $n = 1$. (b) $n = 2$. (c) $n = 3$. (d) $n = 4$.

Let $\mathcal{S}_{n,\beta}$ be an undirected graph with q^n vertices representing the q^n possible cell states, where there is an edge between two vertices $(\ell_1, \ell_2, \ldots, \ell_n)$, $(\ell'_1, \ell'_2, \ldots, \ell'_n) \in \{0, \ldots, q-1\}^n$ if and only if $(|\ell_1 - \ell'_1|, |\ell_2 - \ell'_2|, \ldots, |\ell_n - \ell'_n|) \in M_{n,\beta}$. Examples of $\mathcal{S}_{n,\beta}$ for $n = 1, 2, 3, 4$ are shown in Fig. 9. The space-constrained code is a mapping from the vertices of $\mathcal{S}_{n,\beta}$ to data values $\{0, 1, \ldots, v-1\}$, and the edges show the allowed cell-state transitions via a rewrite. We can see that the degree of the graph $|M_{n,\beta}|$ gives an upper bound for $v - 1$, while the size of the maximum clique in $\mathcal{S}_{n,\beta}$ gives a lower bound for the maximum value of v.

CHAPTER V

CONCLUSION

In this work, we study coding techniques for flash memories and PCMs based on their unique properties. The topics we have studied include error scrubbing codes, optimized cell programming schemes for flash memories, and constrained codes for recording data in PCMs. For systems using flash memories or PCMs, our solutions to these problems can extend their longevity, and improve their reliability and performance.

We have introduced the concept of error scrubbing codes for memory scrubbing in multi-level flash memories without using block erasures. We have presented two code constructions, i.e., the linear error scrubbing codes and modular error scrubbing codes. It is shown that error-scrubbing codes can outperform conventional error-correcting codes because through the actively adjustment of cell levels, the decoding sphere size can be minimized. In this area, an open problem is how to design efficient error scrubbing codes of high rates and with the capability to correct numerous errors.

We have studied methods to program flash-memory cells accurately. Based on the iterative and monotonic cell-programming method, a cell-programming strategy is presented for two performance metrics, which are suitable for the multi-level cell technology and the rank modulation technology, respectively. We have presented an effective algorithm for finding the optimal programming strategy. There are many ways to extend the results, including considering more general programming noise models, studying the joint programming of flash memory cells, and designing coding schemes that approach the storage capacity.

We have considered thermal interference problems for PCMs, which can be challenging when the cell density scales toward its limit. We have considered the thermal

crosstalk problem and the local thermal accumulation problem, and proposed new constrained codes for solving them. We have studied the capacity of the constrained codes and some code constructions. For the symbol-constrained codes, the capacity and code construction for small γ (corresponding to less serious crosstalk between cells) still need to be studied. For space-time constrained codes, the capacity and construction of codes with both space and time constraints still need to be understood. How to combine constrained codes and error-correcting codes for PCMs is still a very interesting unsolved problem. They all remain as open questions.

REFERENCES

[1] A. Bandyopadhyay, G. Serrano and P. Hasler, "Programming analog computational memory elements to 0.2% accuracy over 3.5 decades using a predictive method," in *Proc. IEEE International Symposium on Circuits and Systems*, May 2005, pp. 2148–2151, IEEE, Kobe, Japan.

[2] V. Bohossian, A. Jiang and J. Bruck, "Buffer coding for asymmetric multilevel memory," in *Proc. IEEE International Symposium on Information Theory (ISIT)*, June 2007, pp. 1186–1190, IEEE, Nice, France.

[3] G. W. Burr, "Phase change memory technology," to appear in *Journal of Vacuum Science and Technology*, 2010.

[4] P. Cappelletti, C. Golla, P. Olivo and E. Zanoni, Ed., *Flash memories*, 1st Edition, Boston: Kluwer Academic Publishers, 1999.

[5] Y. Cassuto, M. Schwartz, V. Bohossian and J. Bruck, "Codes for multilevel flash memories: Correcting asymmetric limited-magnitude errors," in *Proc. IEEE International Symposium on Information Theory (ISIT)*, June 2007, pp. 1176–1180, IEEE, Nice, France.

[6] Electronics Industry Market Research and Knowledge Network, "2008 Flash Memory Market Report," http://www.electronics.ca/reports/ic/memory_flash.html, accessed on June 15, 2009.

[7] A. Fiat and A. Shamir, "Generalized write-once memories," *IEEE Transactions on Information Theory*, vol. 30, no. 3, pp. 470–480, May 1984.

[8] F. Fu and A. J. Han Vinck, "On the capacity of generalized write-once memory with state transitions described by an arbitrary directed acyclic graph," *IEEE Transactions on Information Theory*, vol. 45, no. 1, pp. 308–313, 1999.

[9] A. Jiang, V. Bohossian and J. Bruck, "Floating codes for joint information storage in write asymmetric memories," in *Proc. IEEE International Symposium on Information Theory (ISIT)*, June 2007, pp. 1166–1170, IEEE, Nice, France.

[10] A. Jiang and J. Bruck, "Joint coding for flash memory storage," in *Proc. IEEE International Symposium on Information Theory (ISIT)*, July 2008, pp. 1741–1745, IEEE, Toronto, Canada.

[11] A. Jiang and J. Bruck, "On the capacity of flash memories," in *Proc. International Symposium on Information Theory and Its Applications (ISITA)*, December 2008, pp. 94–99, IEEE, Auckland, New Zealand.

[12] A. Jiang, M. Langberg, M. Schwartz and J. Bruck, "Universal rewriting in constrained memories," in *Proc. IEEE International Symposium on Information Theory (ISIT)*, June 2009, pp. 1219–1223, IEEE, Seoul, Korea.

[13] A. Jiang, R. Mateescu, M. Schwartz and J. Bruck, "Rank modulation for flash memories," in *Proc. IEEE International Symposium on Information Theory (ISIT)*, July 2008, pp. 1731–1735, IEEE, Toronto, Canada.

[14] A. Jiang, R. Mateescu, M. Schwartz and J. Bruck, "Rank modulation for flash memories," in *IEEE Transactions on Information Theory*, vol. 55, no. 6, pp. 2659–2673, June 2009.

[15] A. Jiang, M. Schwartz and J. Bruck, "Error-correcting codes for rank modulation," in *Proc. IEEE International Symposium on Information Theory (ISIT)*,

July 2008, pp. 1736–1740, IEEE, Toronto, Canada.

[16] L. A. Lastras-Montano, M. Franceschini , T. Mittelholzer, J. Karidis and M. Wegman, "On the lifetime of multilevel memories," in *Proc. IEEE International Symposium on Information Theory (ISIT)*, June 2009, pp. 1224–1228, IEEE, Seoul, Korea.

[17] S. Lin and D. J. Costello, *Error control coding: fundamentals and applications*, 2nd edition, Upper Saddle River, N.J.: Pearson-Prentice Hall, 2004.

[18] B. H. Marcus, R. M. Roth and P. H. Siegel, *An introduction to coding for constrained systems*, 5th Edition, online: http://www.math.ubc.ca/marcus/Handbook/index.html, accessed on May 1, 2010.

[19] A. Pirovano, A. Redaelli, F. Pellizzer, F. Ottogalli, M. Tosi, D. Ielmini, AL Lacaita, and R. Bez, "Reliability study of phase-change nonvolatile memories," in *IEEE Transactions on Device and Materials Reliability*, vol. 4, no. 3, pp. 422–427, September 2004.

[20] V. Pless, *Introduction to the Theory of Error-Correcting Codes*, 3rd Edition, New York: Wiley, 1998.

[21] T. Richardson and R. Urbanke, *Modern Coding Theory*, 1st Edition, New York: Cambridge University Press, 2008.

[22] R. L. Rivest and A. Shamir, "How to reuse a 'write-once' memory," in *Information and Control*, vol. 55, pp. 1–19, 1982.

[23] A. M. Saleh, J. J. Serrano and J. H. Patel, "Reliability of scrubbing recovery-techniques for memory systems," in *IEEE Transactions on Reliability*, vol. 39,

no. 1, pp. 114–122, 1990.

VITA

Hao Li received his Bachelor of Science degree in computer science from Tsinghua University in 1998. He received his Master of Science degree from Chinese Academy of Sciences in 2001. His research interests include coding theory and storage techniques in flash memories.

Hao Li may be reached at Department of Computer Science and Engineering, c/o Dr. Anxiao Jiang, Texas A&M University, College Station, TX 77843-3128. His email is hockfly@gmail.com.

The typist for this thesis was Hao Li.